

CPT Meeting Room Scheduler

Titolo del progetto: CPT Meeting Room Scheduler

Alunno/a: Luca Di Bello

Classe: Info 4

Anno scolastico: 2019/2020

Docente responsabile: Fabrizio Valsangiacomo

1	Introduzione.....	4
1.1	Informazioni sul progetto.....	4
1.2	Abstract.....	4
1.3	Scopo.....	4
2	Analisi.....	5
2.1	Analisi del dominio.....	5
2.2	Analisi e specifica dei requisiti.....	5
2.3	Use case.....	9
2.4	Pianificazione.....	10
2.4.1	Spiegazione diagramma Gantt.....	11
2.5	Analisi dei mezzi.....	13
2.5.1	Software.....	13
2.5.2	Hardware.....	13
3	Progettazione.....	14
3.1	Design dell'architettura del sistema.....	14
3.2	Design dei dati e database.....	15
3.2.1	Schema logico.....	16
3.2.2	Tipi di dati.....	16
3.3	Design delle interfacce.....	18
3.4	Design procedurale.....	20
4	Implementazione.....	21
4.1	Diagrammi UML.....	21
4.1.1	Controllers.....	21
4.1.2	Models.....	22
4.1.3	Librerie.....	23
4.2	Gestione permessi.....	24
4.2.1	Classe PermissionModel e classe Permissions.....	25
4.2.2	Controllo dei permessi.....	26
4.3	Pannello admin.....	27
4.3.1	Gestione utenti.....	28
4.4	Calendario.....	39
4.4.1	Lettura dati.....	41
4.4.2	Informazioni ed eliminazione di una prenotazione.....	42

4.4.3	Inserimento di prenotazioni.....	43
4.4.4	Modifica data ed ora di un evento.....	44
4.4.5	Notifica errori.....	45
4.4.6	Refresh calendario.....	45
4.4.7	Validazione prenotazioni.....	46
4.5	Generazione di report.....	49
4.6	Gestione notifiche.....	56
4.7	Struttura API.....	57
4.8	Esempio di API.....	58
4.8.1	API User.....	59
4.8.2	API Booking.....	61
4.8.3	API Calendar.....	62
5	Test.....	68
5.1	Protocollo di test.....	68
5.2	Risultati test.....	71
5.3	Mancanze/limitazioni conosciute.....	71
6	Consuntivo.....	68
7	Conclusioni.....	69
7.1	Sviluppi futuri.....	70
7.2	Considerazioni personali.....	70
8	Bibliografia.....	71
8.1	Sitografia.....	71
9	Allegati.....	71

Introduzione

1.1 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

1.2 Abstract

The meeting room has been booked for many years with a table printed on a sheet of paper, this approach is not very safe as a sheet of paper is easily lost or damaged.

With my product you can finally replace the sheet of paper with a more secure, functional and accessible computer system: in fact, with this system you can book directly with your smartphone or computer from your office or home.

All bookings are saved in a secure database and can be modified via an interactive calendar directly from your internet browser.

With this complex system you can schedule a meeting from any device, anywhere, anytime.

1.3 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.

2 Analisi

2.1 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.

2.2 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 prenotazioni 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 <i>active directory</i> di scuola
Sotto requisiti	
001	Pagina web con form che permette all'utente di inserire le proprie credenziali, quindi e-mail e password
002	Soltanto i professori possono accedere al servizio (...@edu.ti.ch)
003	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 permessi 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.

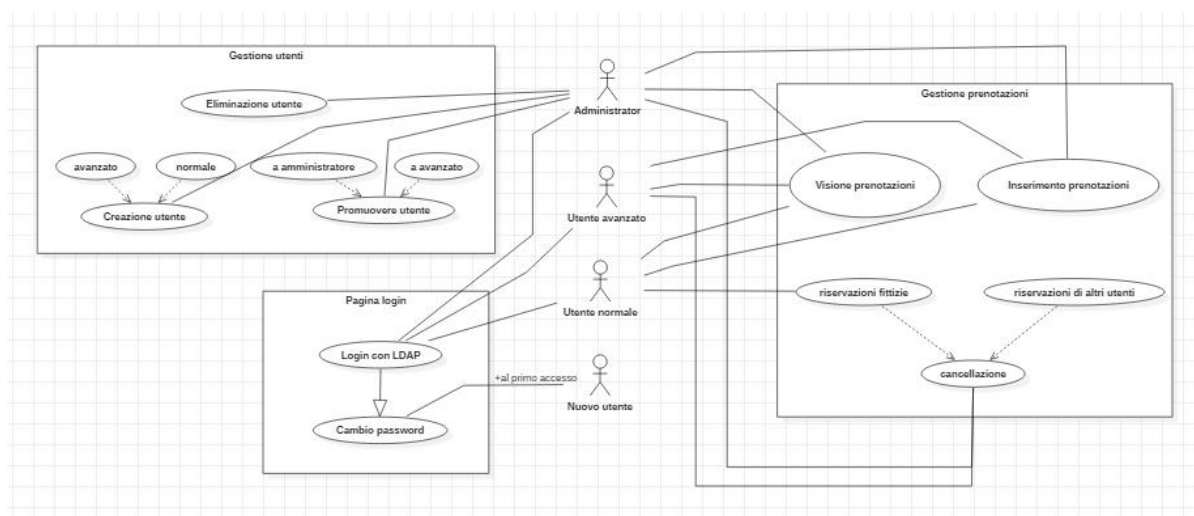


Figure 1 - Diagramma Use Case

2.3 Use case

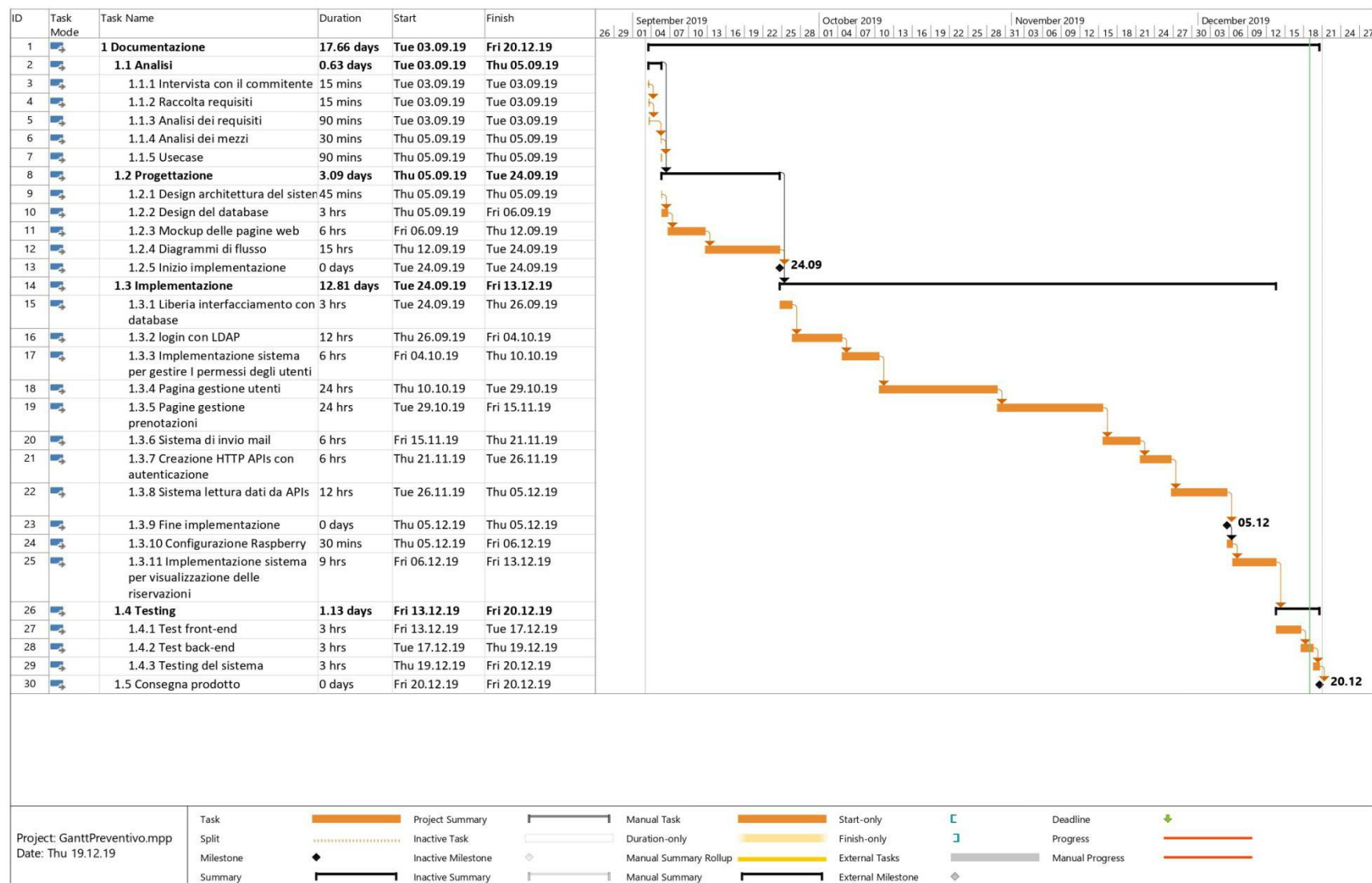
Come si può vedere molto bene dall'immagine sovrastante ho deciso di dividere il diagramma in 3 macro-gruppi (gestione utenti, gestione prenotazioni e pagina login) per semplificare sia la lettura che la stesura del diagramma.

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".

2.4 Pianificazione



2.4.1 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.

2.4.1.1 Analisi

1.1 Analisi	0,63 days	Tue 03/09/19	Thu 05/09/19
1.1.1 Intervista con il committente	15 mins	Tue 03/09/19	Tue 03/09/19
1.1.2 Raccolta requisiti	15 mins	Tue 03/09/19	Tue 03/09/19
1.1.3 Analisi dei requisiti	90 mins	Tue 03/09/19	Tue 03/09/19
1.1.4 Analisi dei mezzi	30 mins	Thu 05/09/19	Thu 05/09/19
1.1.5 Usecase	90 mins	Thu 05/09/19	Thu 05/09/19

Figure 3 - Gantt attività analisi

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

Intervista con il committente

- o Intervista con il committente, utilizzata per chiarire i dubbi relativi alla consegna

Raccolta dei requisiti

- o Raccolta dei requisiti tramite la lettura approfondita del QdC (Quaderno Dei Compiti)

Analisi dei requisiti

- o Riflessione riguardo alle richieste raccolte, analizzandole e suddividendole in modo sensato

Analisi dei mezzi

- o Riflessione riguardo ai mezzi da utilizzare per il progetto (sia Software che Hardware)

Use case

- o Stesura del diagramma Use Case

1.2 Progettazione	3,22 days	Thu 05/09/19
1.2.1 Design architettura del sistema	45 mins	Thu 05/09/19
1.2.2 Design del database	6 hrs	Thu 05/09/19
1.2.3 Mockup delle pagine web	4 hrs	Tue 10/09/19
1.2.4 Diagrammi di flusso	15 hrs	Fri 13/09/19

Figure 4 - Gantt attività progettazione

2.4.1.2 Progettazione

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

Design architettura del sistema

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

1.3 Implementazione	12,38 days	Thu 26/09/19
1.3.1 Libreria interfacciamento con database	6 hrs	Thu 26/09/19
1.3.2 login con LDAP	12 hrs	Tue 01/10/19
1.3.3 Implementazione sistema per gestire i permessi degli utenti	9 hrs	Thu 10/10/19
1.3.4 Pagina gestione utenti	24 hrs	Thu 17/10/19
1.3.5 Pagine gestione prenotazioni	24 hrs	Tue 05/11/19
1.3.6 Sistema di invio mail	6 hrs	Fri 22/11/19
1.3.7 Creazione HTTP APIs con autenticazione	6 hrs	Thu 28/11/19
1.3.8 Sistema lettura dati da APIs	12 hrs	Tue 03/12/19

Figure 5 - Gantt attività implementazione

2.4.1.3 Implementazione

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

- Libreria interfacciamento con database
 - o Implementazione di una libreria che permetterà la connessione e l'invio di query al database
- Login con LDAP
 - o Sistema di login utilizzando LDAP.
- Implementazione sistema per gestire i permessi degli utenti
 - o Implementazione di un sistema che controlla i permessi di un utente prima di eseguire qualsiasi operazione
- Pagina gestione utenti
 - o Pagina che permette di gestire gli utenti come da requisiti
- Pagina gestione prenotazioni
 - o Pagina che permette di gestire le prenotazioni come da requisiti
- Sistema di invio mail
 - o Sistema che permette l'invio di mail di conferma ad un determinato utente
- Creazione http APIs con autenticazione
 - o 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
 - o Sistema di lettura e formattazione dei dati utilizzando le API fornite dal sito web

2.4.1.4 Testing

1.4 Testing	1,5 days	Thu 12/12/19
1.4.1 Test front-end	3 hrs	Thu 12/12/19
1.4.2 Test back-end	3 hrs	Fri 13/12/19
1.4.3 Testing del sistema	6 hrs	Tue 17/12/19

Figure 6 - Gantt attività testing

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

Test front-end

- o Testing delle pagine web nella loro interezza

Test back-end

- o Testing di tutto il back-end

Testing del sistema

- o Testing approfondito dell'intero sistema

2.5 Analisi dei mezzi

2.5.1 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

MDB 4.8.3: Acronimo di *Material Design for Bootstrap*. Esso è un framework front-end basato su *Bootstrap 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

2.5.2 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):

- o Windows 10 Home versione 1809
- o 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 Desktop

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

3 Progettazione

3.1 Design dell'architettura del sistema

Lo schema sovrastante è composto da 8 parti principali collegate tra loro in questo modo: Il sito

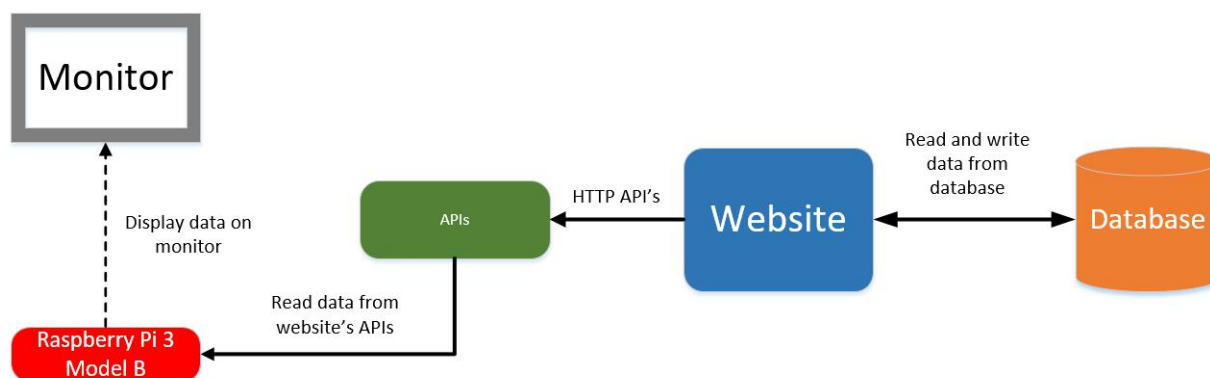


Figure 7 - Struttura del programma

web si interfacerà 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 (sotto forma di JSON) i quali verranno parsati (formattati) e mostrati all'utente finale tramite un Monitor.

3.2 Design dei dati e database

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

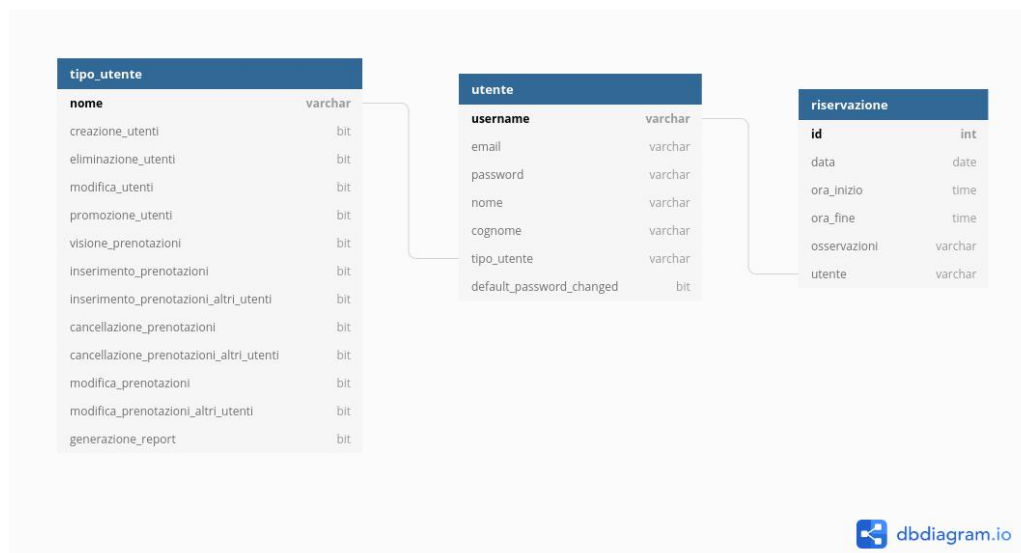


Figure 8 - Diagramma E/R database

Lo schema è composto di 3 entità principali, una delle quali non è altro che una generalizzazione (*tipo_utente*).

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.

(Nota: il diagramma è stato creato tramite un applicativo web chiamato *dbdiagram.io*)

3.2.1 Schema logico

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

Tipo_utente(nome, creazione_utenti, eliminazione_utenti, promozione_utenti, visione_prenotazioni, inserimento_prenotazioni, inserimento_prenotazioni_altri_utenti, cancellazione_prenotazioni, cancellazione_prenotazioni_altri_utenti, modifica_prenotazioni, modifica_prenotazioni_altri_utenti, generazione_report)

Utente(username, email, password, nome, cognome, tipo_utente (FK), default_password_changed)

Riservazione(id, data, ora_inizio, ora_fine, osservazioni, utente (FK))

3.2.2 Tipi di dati

Tipo_Utente	
Nome attributo	Tipo attributo
Nome	<i>Primary Key, Varchar(50)</i>
Creazione_utenti	<i>Bit, default 0</i>
Eliminazione_utenti	<i>Bit, default 0</i>
Promozione_utenti	<i>Bit, default 0</i>
Visione_prenotazioni	<i>Bit, default 0</i>
Inserimento_prenotazioni	<i>Bit, default 0</i>
Inserimento_prenotazioni_altri_utenti	<i>Bit, default 0</i>
Cancellazione_prenotazioni	<i>Bit, default 0</i>
Cancellazione_prenotazioni_altri_utenti	<i>Bit, default 0</i>
Modifica_prenotazioni	<i>Bit, default 0</i>
Modifica_prenotazioni_altri_utenti	<i>Bit, default 0</i>
Generazione_report	<i>Bit, default 0</i>

Utente	
Nome attributo	Tipo attributo
Username	<i>Primary Key, Varchar(255)</i>
Email	<i>Varchar(255)</i>
Password	<i>Varchar(255)</i>
Nome	<i>Varchar(100)</i>
Cognome	<i>Varchar(100)</i>
Tipo_utente	<i>Foreign Key, Varchar(50)</i>
Default_password_changed	<i>Bit, default 0</i>

Riservazione	
Nome attributo	Tipo attributo
Id	<i>Primary Key, Int(11), Auto Increment</i>
Date	<i>Date</i>
Ora_inizio	<i>Time</i>
Ora_fine	<i>Time</i>
Osservazioni	<i>Varchar(512)</i>
Utente	<i>Foreign Key, varchar(255)</i>

3.3 Design delle interfacce

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

Ho realizzato 4 mockup utilizzando 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:

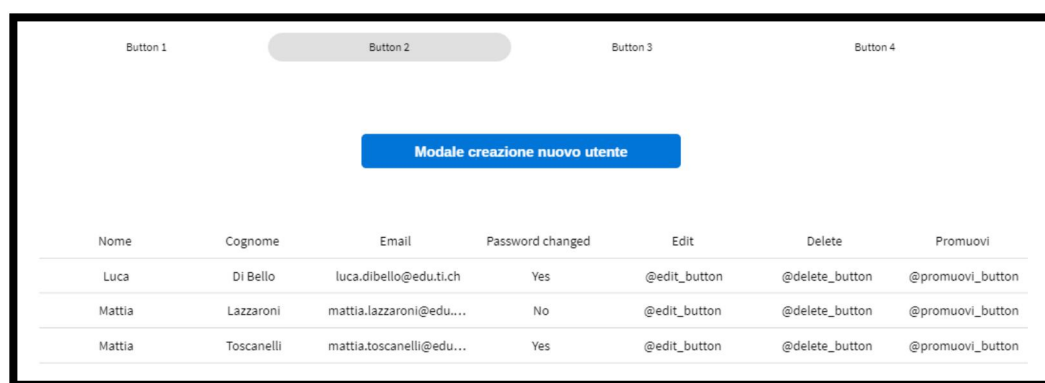
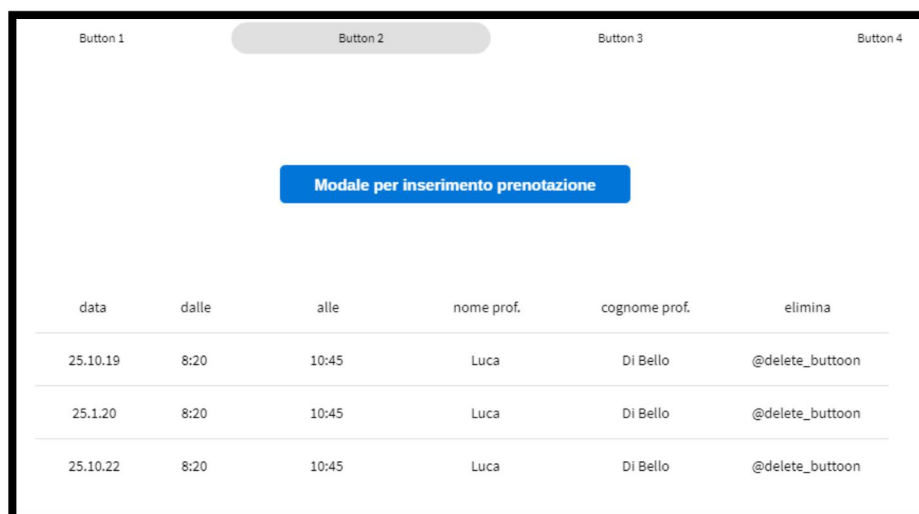


Figure 9 - Pagina gestione utenti

Questa pagina mostra in una tabella tutte le informazioni sugli utenti e possiede anche dei pulsanti utili per eseguire la modifica, eliminazione o la promozione di un utente.

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

Questa pagina invece mostra tutte le informazioni di ogni riservazione. Le informazioni sono mostrate anche qui all'interno di una tabella. Su ogni riga è presente un pulsante utile per eliminare una specifica prenotazione.



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

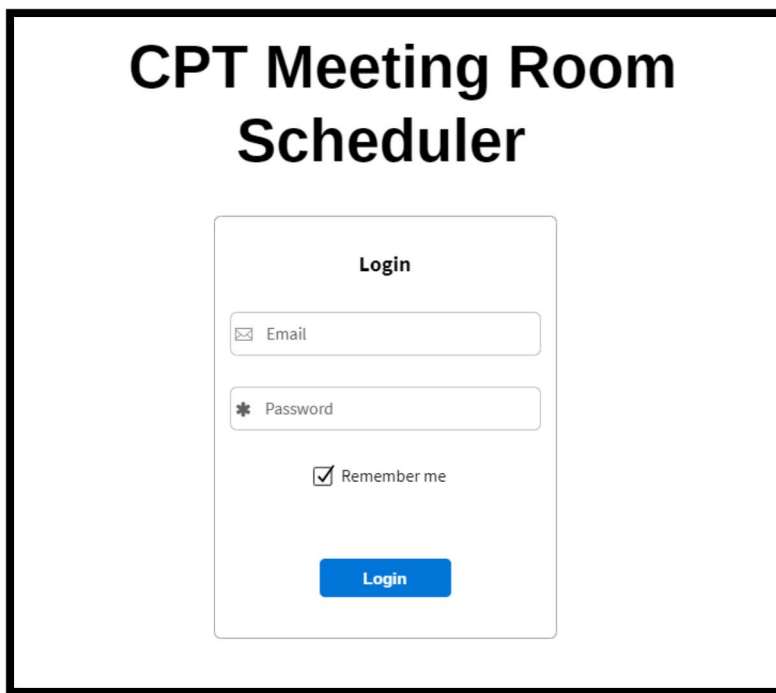



Figure 10 - Pagina di login


Questa semplice pagina raffigura la schermata di login che utilizzeranno gli utenti per accedere al sistema.

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



data	dalle	alle	nome prof.	cognome prof.
25.10.19	8:20	10:45	Luca	Di Bello
25.1.20	8:20	10:45	Luca	Di Bello
25.10.22	8:20	10:45	Luca	Di Bello

Figure 11 - Visualizzazione prenotazioni su monitor

	SAMT - Sezione Informatica	Pagina 20 di 75
	CPT Meeting Room Scheduler	

Come si può vedere questa pagina è molto simile a quella della gestione delle prenotazioni.

3.4 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.

4 Implementazione

4.1 Diagrammi UML

Nei vari sottocapitoli verranno illustrati i diagrammi UML di ogni “sezione” del progetto. Queste sono le varie sezioni del progetto dove ho ritenuto necessaria la creazione di un diagramma UML:

Controllers: tutti i controller che gestiscono le pagine

Models: tutte le classi model che eseguono operazioni sui dati per i controller

Libraries: librerie scritte appositamente per questo progetto

4.1.1 Controllers

Questo è il diagramma UML dei controller presenti all'interno dell'applicativo:

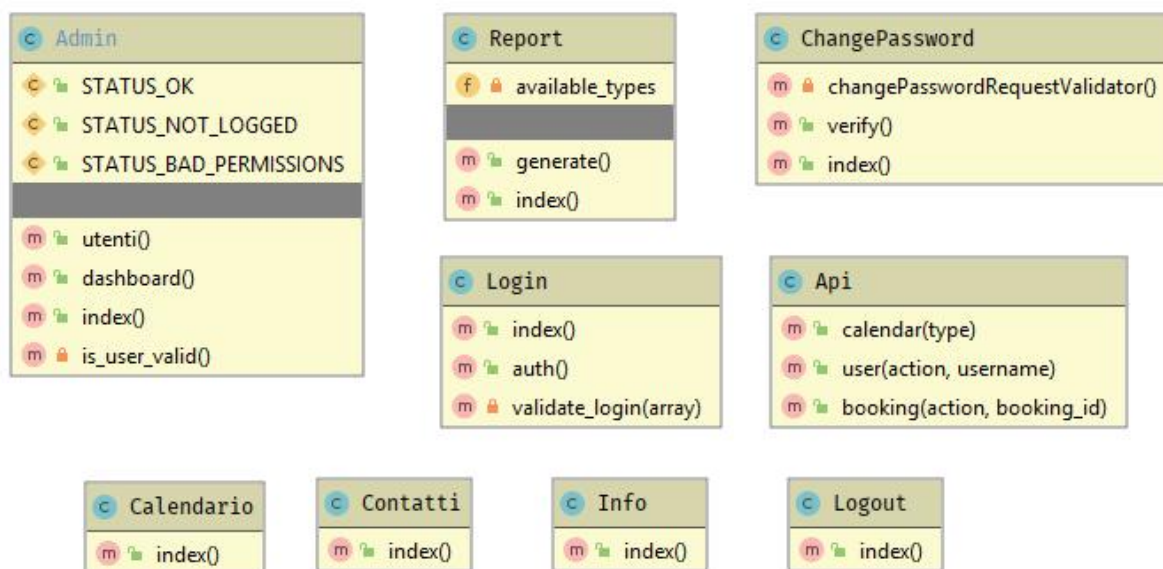


Figure 12 - Diagramma UML controllers

4.1.2 Models

Questo è il diagramma UML dei vari model presenti nel sistema:

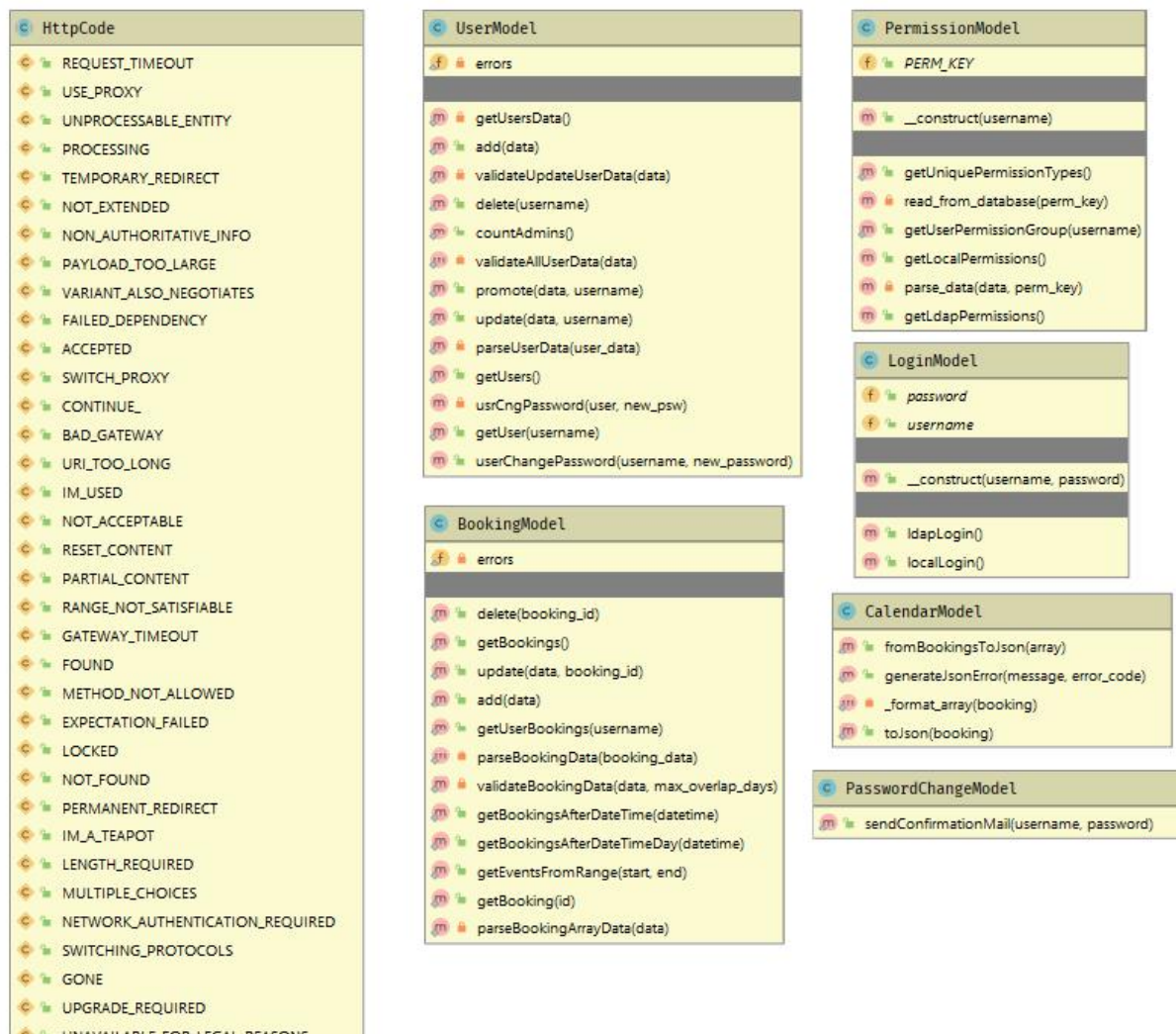


Figure 13 - Diagramma UML models

(Nota: la classe HttpCode è stata troncata in quanto troppo lunga)

4.1.3 Librerie

Questo è il diagramma UML di tutte le librerie scritte da me ed utilizzate all'interno del sistema:

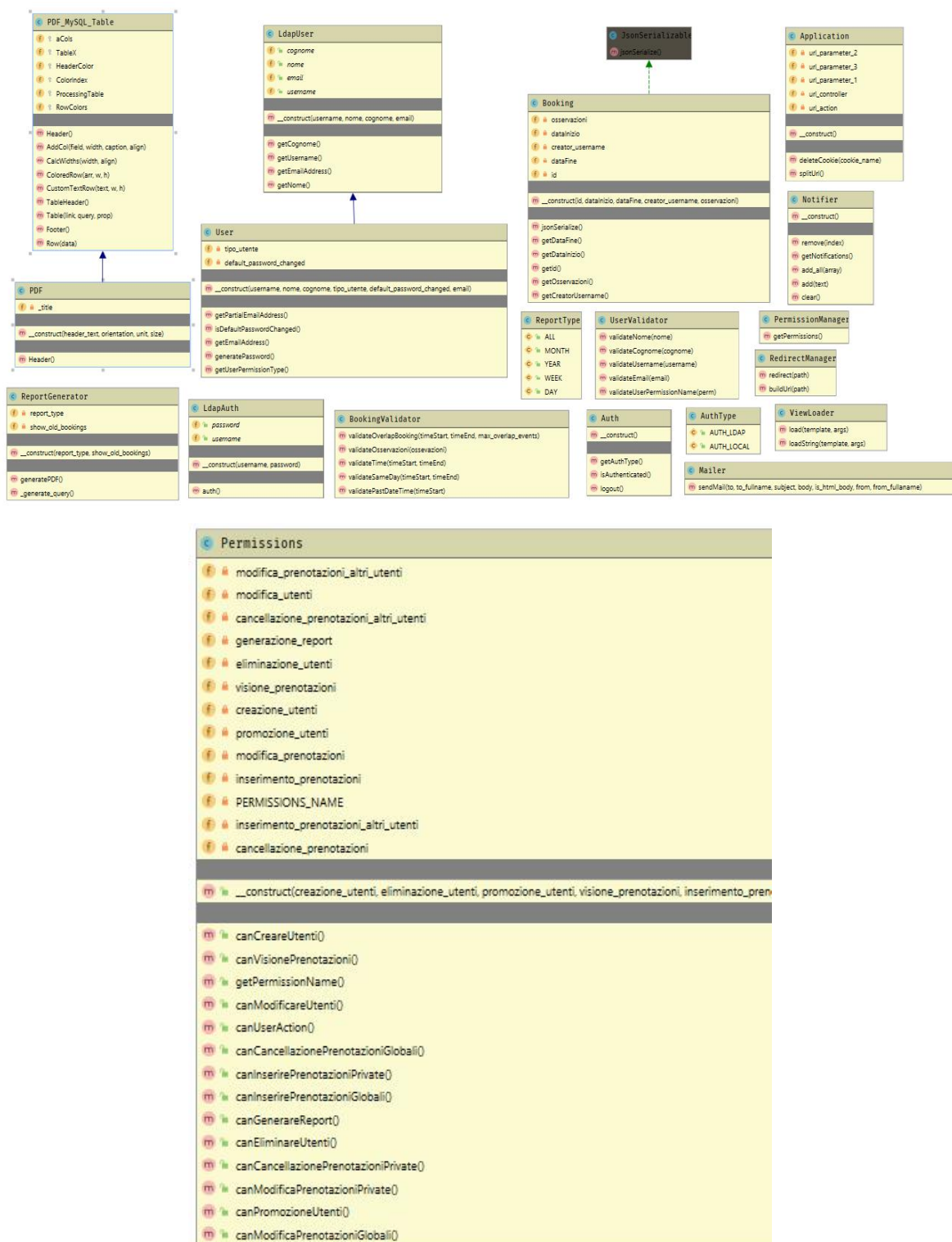


Figure 14 - Diagramma UML librerie

4.2 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.

Questi sono tutti i permessi applicabili agli utenti dell'applicazione:

creazione_utenti

- o Se abilitato l'utente può creare nuovi utenti

eliminazione_utenti

- o Se abilitato l'utente può eliminare altri utenti

modifica_utenti

- o Se abilitato l'utente può modificare altri utenti

promozione_utenti

- o Se abilitato l'utente può promuoverne altri (cambiare i permessi)

visione_prenotazioni

- o Se abilitato l'utente può vedere le prenotazioni

inserimento_prenotazioni

- o Se abilitato l'utente può inserire delle prenotazioni all'interno del calendario

inserimento_prenotazioni_altri_utenti

- o Se abilitato l'utente può inserire prenotazioni a nome di altri utenti

cancellazione_prenotazioni

- o Se abilitato l'utente può eliminare le proprie prenotazioni

cancellazione_prenotazioni_altri_utenti

- o Se abilitato l'utente può eliminare prenotazioni create da altri utenti

modifica_prenotazioni

- o Se abilitato l'utente può modificare le proprie prenotazioni

modifica_prenotazioni_altri_utenti

- o Se abilitato l'utente può modificare prenotazioni di altri utenti

generazione_prenotazioni

- o Se abilitato l'utente può generare file di report

4.2.1 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.

4.2.1.1 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);

}
```

4.2.1.2 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.

4.2.1.3 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").

4.2.1.4 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*.

4.2.2 Controllo dei permessi

Per certe funzioni c'è la necessità di controllare i permessi dell'utente che esegue la richiesta. Questo si può fare utilizzando la classe *PermissionManager*, la quale non è altro che una classe *wrapper* che permette di leggere i permessi dell'utente. Una volta letti i permessi è possibile fare i vari controlli di cosa può e non può fare l'utente. Per esempio questo è un pezzo di codice che controlla se l'utente ha i permessi o meno di eliminarne un altro:

```
if (PermissionManager::getPermissions()->canEliminareUtenti()) {
    $result = UserModel::delete($username);

    if (is_array($result)) {
        $GLOBALS["NOTIFIER"]->add_all($result);
    }
} else {
    $GLOBALS["NOTIFIER"]->add("Non hai i permessi necessari per eliminare gli utenti");
}
```

Questi sono tutti i controlli disponibili:

```

m  getPermissionName(): string
m  canModificareUtenti(): bool
m  canVisionePrenotazioni(): bool
m  canInserirePrenotazioniPrivate(): bool
m  canInserirePrenotazioniGlobali(): bool
m  canCancellazionePrenotazioniPrivate(): bool
m  canCancellazionePrenotazioniGlobali(): bool
m  canModificaPrenotazioniPrivate(): bool
m  canModificaPrenotazioniGlobali(): bool
m  canEliminareUtenti(): bool
m  canCreareUtenti(): bool
m  canPromozioneUtenti(): bool
m  canGenerareReport(): bool
m  canUserAction(): bool

```

Figure 15 - Metodi getter classe Permission

4.3 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.

4.3.1 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*).

Se l'utente entra nel pannello admin ma non dispone nessun permesso relativo alla gestione degli utenti il collegamento alla pagina, il quale solitamente è nella barra di navigazione in cima allo schermo, viene nascosto. Se l'utente prova ad accedere alla pagina il sistema mostra una pagina di errore che fa capire all'utente di non avere i permessi necessari per eseguire alcuna operazione sugli utenti.

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.

4.3.1.1 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_\(XSS\)](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:

```


m validateUsername(username: string): bool
m validateNome(nome: string): bool
m validateCognome(cognome: string): bool
m validateEmail(email: string): bool
m validateUserPermissionName(perm: string): bool

```

Figure 16 - Metodi classe UserValidator

Spiegazione dettagliata del funzionamento di ogni metodo di validazione:

● *validateUsername*

	SAMT - Sezione Informatica	Pagina 29 di 75
	CPT Meeting Room Scheduler	

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_alpha* 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@**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*.

4.3.1.2 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:

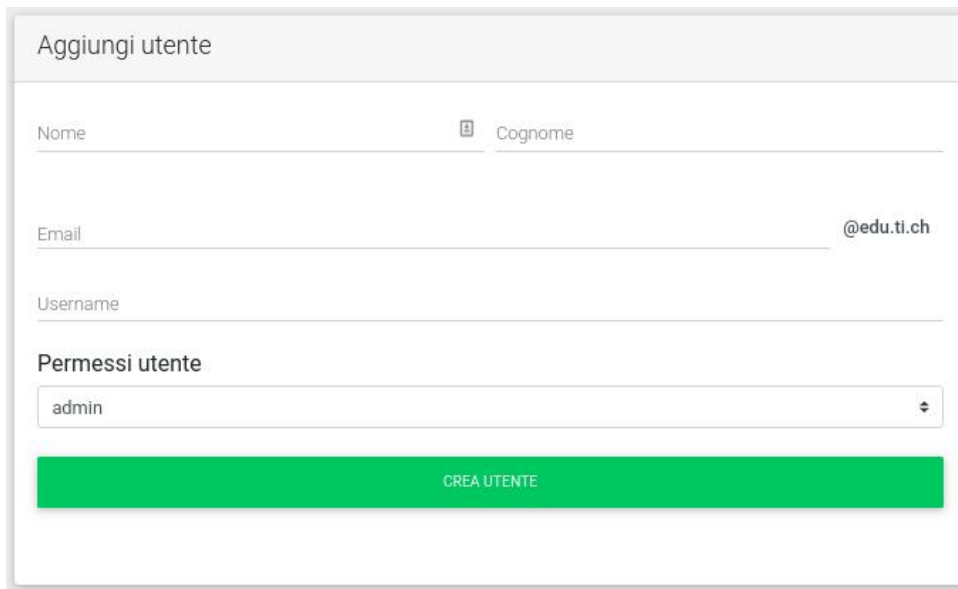


Figure 17 - Maschera creazione utente

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**@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).

Se, per esempio, l'utente *luca.dibello* non dispone dei permessi necessari per la creazione di nuovi utenti l'intero form necessario per la creazione dell'utente viene nascosto.

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:

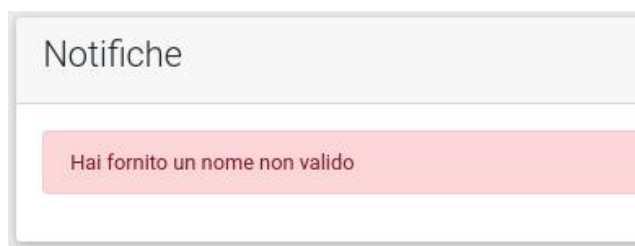


Figure 18 - Esempio notifica di errore

Dopo la creazione di un utente viene inviata automaticamente un email all'indirizzo mail dell'account creato. All'interno di questa mail ci sono scritte le credenziali di accesso per il sistema. Questo è un esempio di email:

Riservazioni Saletta noreply@riservazioni-cpt.ch via_samtinfo.ch
to me ▾

🇮🇹 Italian ▾ > English ▾ [Translate message](#)

CPT Meeting Room Scheduler - Creazione account

Il tuo account è stato creato correttamente all'interno del sistema. Queste sono le credenziali di accesso per il tuo account:

Nome utente	luca.prova
Password	\$2y\$10\$08NzC.HVRfcUYfFn0rP9geIkMPGLsIAYYu/Uccf3AG0Y4MJhoNUrm


...

Clicca [qui](#) per accedere alla pagina di login. Al primo accesso l'utente verrà obbligato a cambiare la password di accesso per motivi di sicurezza.

Questa email è stata inviata automaticamente dal sistema.

4.3.1.3 Visione utenti locali

In questa sezione si possono eseguire molteplici operazioni sugli utenti locali presenti nel sistema. Gli utenti vengono mostrati in una tabella generata dinamicamente da PHP, la quale (tramite la libreria JavaScript chiamata DataTables) viene resa una tabella “avanzata” che permette l’esecuzione di filtri ed ordinamento alfabetico. Ecco come vengono mostrati i dati al suo interno:



The screenshot shows a web interface titled 'Utenti'. It features a table with columns: Username, Email, Nome, cognome, Permessi, Password cambiata, and Opzioni. There are three rows of user data. The 'Opzioni' column contains three buttons: a blue button with a person icon, a red button with a trash icon, and a yellow button with a person icon. Above the table, there is a search bar and a 'Show 10 entries' dropdown. Below the table, it says 'Showing 1 to 3 of 3 entries' and has 'Previous', '1', and 'Next' navigation links.

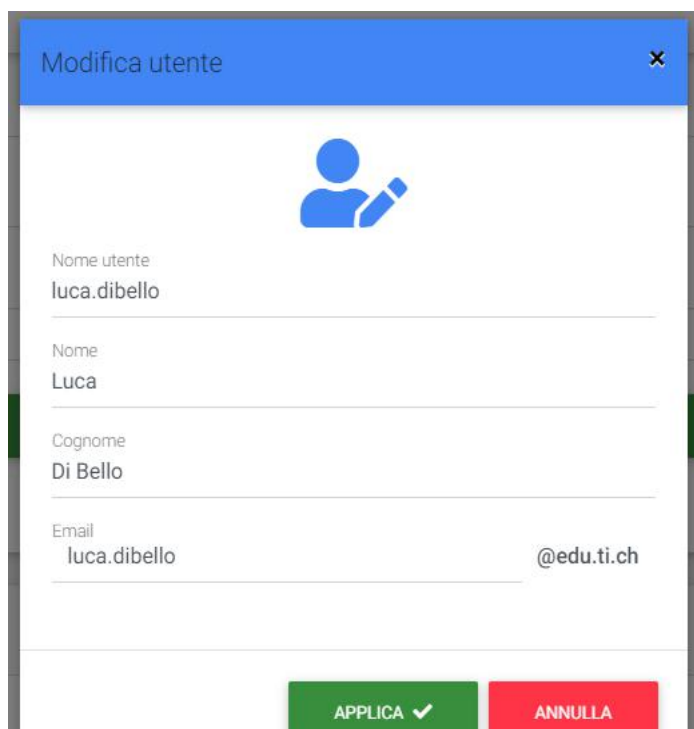
Username	Email	Nome	cognome	Permessi	Password cambiata	Opzioni
luca.dibello	luca.dibello@edu.ti.ch	Luca	Di Bello	admin	Sì	[Blue icon] [Red trash] [Yellow icon]
mattia.lazzaroni	mattia.lazzaroni@edu.ti.ch	Mattia	Lazzaroni	avanzato	Sì	[Blue icon] [Red trash] [Yellow icon]
mattia.toscanelli	mattia.toscanelli@edu.ti.ch	Mattia	Toscanelli	user	Sì	[Blue icon] [Red trash] [Yellow icon]

Figure 19 - Tabella visualizzazione utenti

L’ultima colonna della tabella non contiene dati sugli utenti ma dei pulsanti per eseguire delle operazioni sull’utente desiderato, ecco le loro funzioni:

Modifica utente

Il primo pulsante (quello blu) serve per modificare i dati dell’utente (quindi username, email, nome o cognome). Se viene cliccato viene mostrato a schermo un modale che permette la modifica delle varie informazioni:



The screenshot shows a modal window titled 'Modifica utente'. It contains a blue header with a close button. Below the header is a blue icon of a person with a pencil. The form has four input fields: 'Nome utente' (pre-filled with 'luca.dibello'), 'Nome' (pre-filled with 'Luca'), 'Cognome' (pre-filled with 'Di Bello'), and 'Email' (pre-filled with 'luca.dibello' and '@edu.ti.ch'). At the bottom, there are two buttons: a green 'APPLICA' button with a checkmark and a red 'ANNULLA' button.

Figure 20 - Modale modifica utenti

(Nota: I dati vengono inseriti automaticamente all'interno del form tramite un piccolo script JavaScript)

Cliccando sul pulsante "applica" presente nel modale viene eseguita una richiesta POST alle API specifiche per la modifica degli utenti, raggiungibili tramite questo link:

api/user/update/<username_utente>

Per esempio, se voglio modificare l'utente *luca.dibello* devo inviare i dati a questo link:

<http://example.com/api/user/update/luca.dibello>

```
array (size=4)
  'username' => string 'luca.dibello' (length=12)
  'nome' => string 'Luca' (length=4)
  'cognome' => string 'Di Bello' (length=8)
  'email' => string 'luca.dibello' (length=12)
```

I dati vengono formattati ed inviati con questa semplice struttura:

Se le API riescono a validare correttamente i dati secondo i canoni prestabiliti (vedi capitolo *validazione dati* per ulteriori informazioni) e l'utente che esegue la richiesta ha i permessi necessari per la modifica degli utenti, i dati vengono passati alla classe *model* chiamata *UserModel* (metodo *update*) che si occupa di aggiornare i dati all'interno del database MySQL.

Cliccando sul pulsante "Annulla" il modale viene chiuso e vengono puliti tutti i campi.

Eliminazione utente

Cliccando il pulsante rosso viene aperto invece un modale di conferma di eliminazione. Esso si presenta così all'utente:



Figure 21 - Modale conferma eliminazione utente

Ho deciso di inserire un modale di conferma di eliminazione dell'utente in quanto, se viene eliminato l'utente, anche tutte le sue prenotazioni vengono eliminate a cascata per non avere dati orfani all'interno del database.

Cliccando su “Sì”, viene eseguita una richiesta alle API specifiche per l’eliminazione degli utenti, richiamabili tramite questo link: `api/user/delete/<username_utente>`

Per esempio se si vuole eliminare l’utente “luca.dibello” devo richiamare le API tramite questo link:

<http://example.com/api/user/update/luca.dibello>

Se le API riescono a validare correttamente i dati secondo i canoni prestabiliti (vedi capitolo *validazione dati* per ulteriori informazioni) e l’utente che esegue la richiesta ha i permessi necessari per l’eliminazione degli utenti, i dati vengono passati alla classe *model* chiamata *UserModel* (metodo *delete*) che si occupa di aggiornare i dati all’interno del database MySQL.

Se invece si clicca sul pulsante “No” l’operazione viene annullata e quindi i dati non vengono eliminati.

Promozione utente

Cliccando l’ultimo pulsante (quello color ambra) viene aperto un modale che permette di cambiare i permessi ad un determinato utente. Ecco come si presenta l’interfaccia all’utente:

Figure 22 - Modale promozione utente

Esso mostra i permessi attuali dell’utente in un input di tipo testo *read-only* (non modificabile) ed un menù a tendina dove si può scegliere il nuovo permesso da applicare all’utente. I permessi all’interno di questo menù a tendina vengono caricati dinamicamente dal database tramite la classe *model* chiamata *PermissionModel*.

Cliccando sul tasto applica viene inviata una richiesta POST alle API che si occupano del cambio dei permessi di un utente. Esse sono raggiungibili tramite questo link:
`api/user/promote/<username>`

Per esempio, se voglio cambiare i permessi all’utente *luca.dibello* devo inviare i dati a questo link:

<http://example.com/api/user/promote/luca.dibello>

I dati inviati nella richiesta vanno formattati in questo modo (esempio cambio di permessi *admin*):

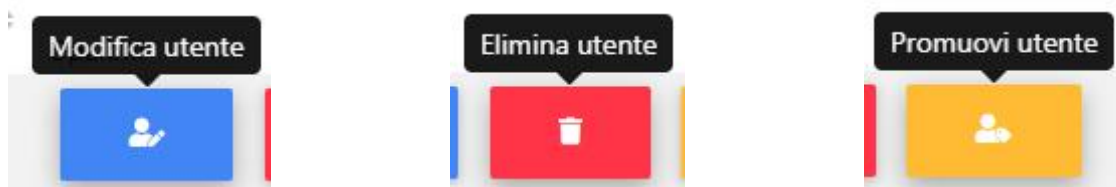
```
array (size=1)
  'tipo_utente' => string 'admin' (length=5)
```

Se le API riescono a validare correttamente i dati secondo i canoni prestabiliti (vedi capitolo *validazione dati* per ulteriori informazioni) e l'utente che esegue la richiesta ha i permessi necessari per la promozione degli utenti, i dati vengono passati alla classe *model* chiamata *UserModel* (metodo *promote*) che si occupa di aggiornare il campo relativo al tipo di permessi assegnato all'utente all'interno del database MySQL.

Se invece si clicca su “*Annulla*” l'operazione viene annullata e nessun dato viene modificato.

(Nota: dato che i permessi vengono caricati al login dell'utente, se si vuole cambiare i permessi all'utente con cui sono loggato in quel momento è necessario eseguire un logout-login per applicare le modifiche)

Tutti e tre i pulsanti presentano dei *tool-tips* che vengono mostrati al passaggio del mouse. Essi servono per permettere all'utente di capire a cosa serve ogni singolo pulsante senza dover per forza cliccarlo e quindi aprire il modale. Ecco come vengono mostrati i *tool-tip* su ogni singolo pulsante:



Questo è stato possibile grazie ad una libreria JavaScript chiamata *popper.js*, la quale è già presente nel framework front-end che ho utilizzato per lo sviluppo di questo progetto (MDBBootstrap).

I pulsanti vengono nascosti automaticamente a seconda dei permessi dell'utente che visualizza la pagina: se per esempio l'utente *luca.dibello* non dispone dei permessi necessari per l'eliminazione degli utenti, il pulsante non viene nemmeno mostrato a schermo. Stessa cosa per tutti e tre i pulsanti.

Se invece l'utente non ha i permessi per la visualizzazione degli utenti presenti nel sistema l'intera tabella non verrà mostrata.

Se durante un'azione di modifica, eliminazione, creazione o promozione di un utente vengono prodotti degli errori, sia di validazione che a livello di database o codice, viene mostrata una notifica nel blocco posto in cima alla pagina chiamato "*Notifiche*".

Se non sono presenti notifiche esso si mostra all'utente in questa maniera:

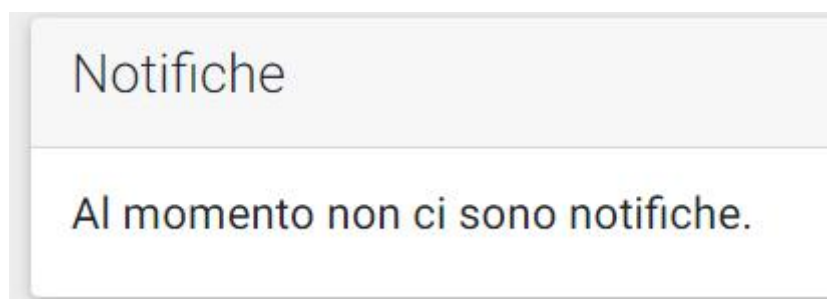


Figure 23 - Notifiche

Mentre, se sono generati degli errori, essi vengono mostrati uno sotto l'altro:



Figure 24 - Esempio di errori nelle notifiche

(Nota: per ulteriori informazioni sul sistema che si occupa della gestione delle notifiche vedi capitolo "*gestione notifiche*")

4.4 Calendario

Ho sviluppato il calendario interattivo utilizzando una libreria molto famosa e versatile chiamata FullCalendar. Questo è il sito web ufficiale della libreria: <https://fullcalendar.io/>.

Il calendario permette all'utente di visualizzare, creare, modificare ed eliminare gli eventi in modo molto semplice e veloce.

Questo è come vengono visualizzati gli eventi all'interno del calendario da un dispositivo desktop (larghezza del monitor maggiore di 1000px):

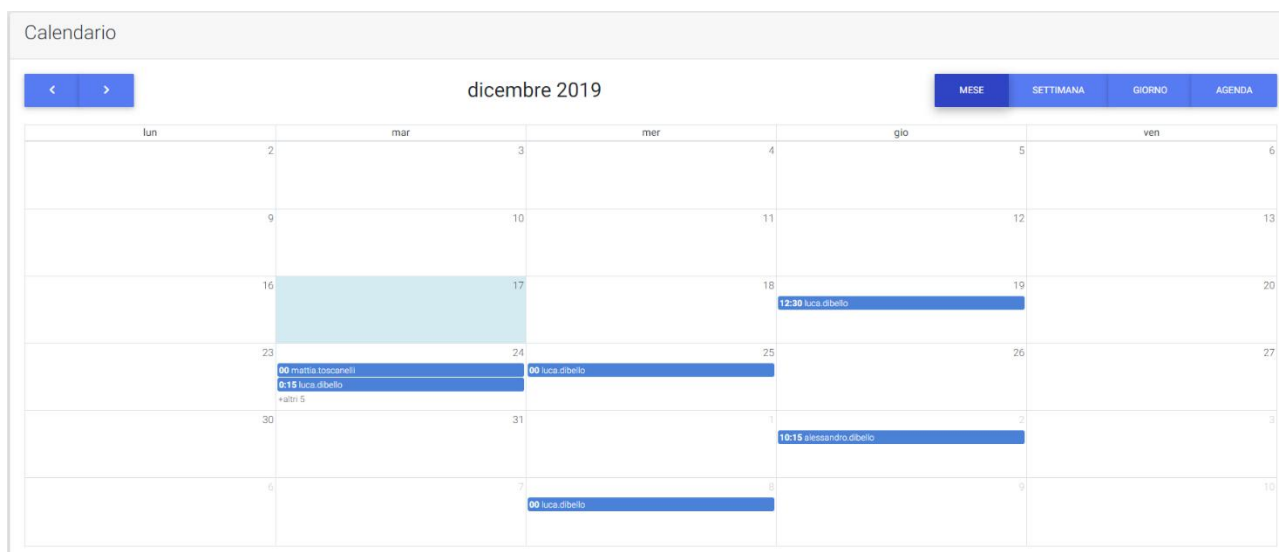


Figure 25 - Calendario interattivo

Invece così e come viene visualizzato il calendario da dispositivi mobili (larghezza dello schermo minore di 1000px):

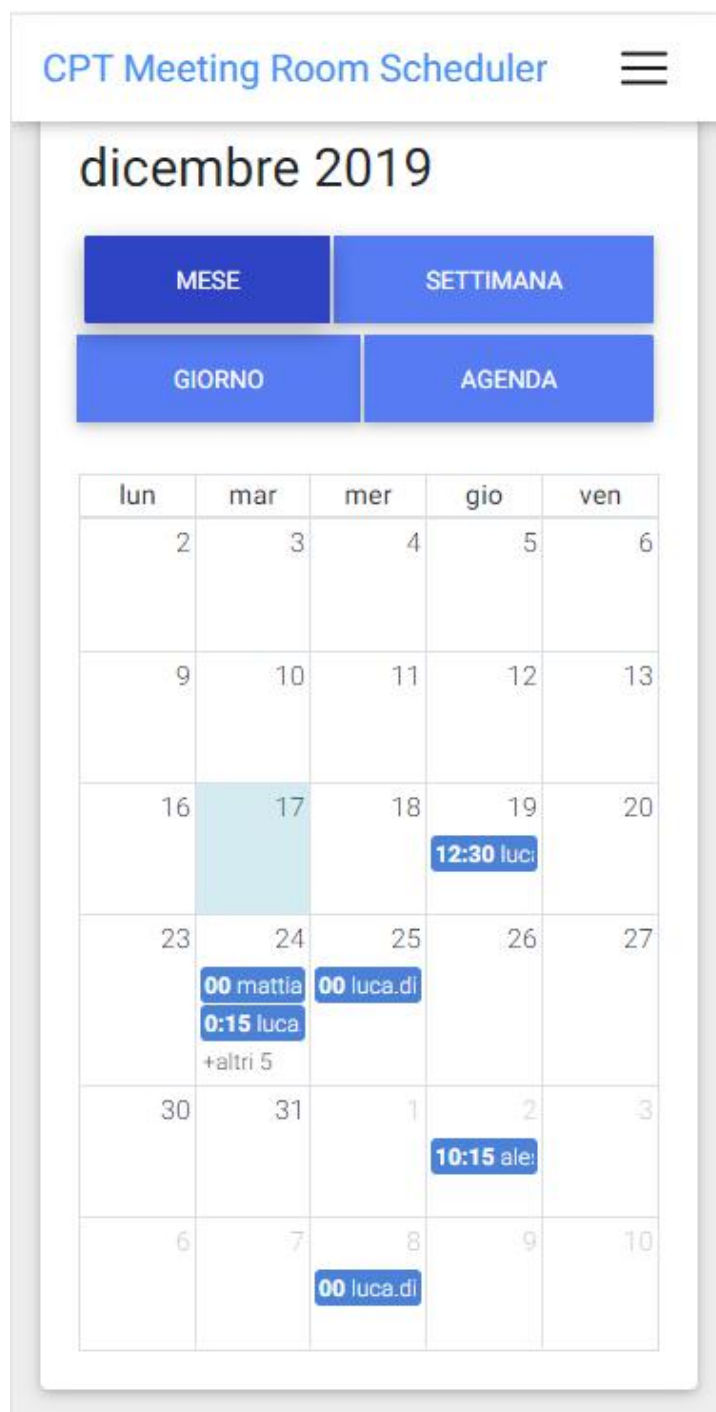


Figure 26 - Calendario interattivo da dispositivo mobile

Inizialmente la visualizzazione da telefono del calendario risultava impossibile dato che non era responsivo. Con qualche regola di CSS applicata con le media query sono riuscito a rendere il calendario leggibile ed utilizzabile da qualsiasi dispositivo.

Per ulteriori informazioni sulle API citate molteplici volte all'interno di questo capitolo leggere il capitolo 4.7 il quale spiega dettagliatamente come sono strutturate e come funzionano le API.

4.4.1 Lettura dati

Dato che il calendario è completamente implementato con JavaScript i dati vengono caricati tramite una richiesta AJAX alle API. Il calendario infatti permette di caricare i dati tramite JSON (sia hard-coded nella pagina che in maniera procedurale). Questa è la funzione del calendario che viene richiamata automaticamente dal calendario per caricare i dati al suo interno:

```
events: function (start, end, callback){...}
```

All'interno di questa funzione viene eseguita una richiesta AJAX alle API del calendario, le quali ritornano un JSON formattato in modo da essere riconosciuto dal calendario. Questo è il codice che esegue la richiesta:

```
$.ajax({
  // Calls the calendar api
  url: 'http://saminfo.ch/riservazioneale2019/api/calendar',
  method: 'POST',
  dataType: 'json',
  data: {
    token: "<API_TOKEN>" // Setup API authentication code
  },
  success: function (doc) {
    // Letto il JSON dalle API
    ...
  }
});
```

Una volta ricevuto il JSON contenente tutti gli eventi da mostrare all'interno del calendario, essi vengono trasformati in un array di dizionari (oggetti *dict*) che permettono al calendario di leggere correttamente i dati e di mostrarli all'interno della griglia:

```
let events = [];

$.map(doc, function (r) {
  // Add event to list
  let name = r.professor.name;
  let surname = r.professor.surname;

  events.push({
    // Build dict
    id: r.id,
    title: r.title,
    start: r.start,
    end: r.end,
    extendedProps: {
      note: r.note,
      professor_name: name,
      professor_surname: surname,
    }
  })
});
```

Questo codice non fa altro che leggere riga per riga il JSON e trasformare pian piano tutte i dati descritti al suo interno in un array di oggetti *dict*.

Una volta creato l'array notifico al calendario che sono riuscito a leggere e *parsare* tutti gli eventi. Questo è possibile farlo tramite una funzione *callback* offerta dalla libreria:

```
// Return events
callback(events);
```

4.4.2 Informazioni ed eliminazione di una prenotazione

La libreria offre un evento che viene richiamato quando l'utente clicca su un evento presente nel calendario. Quando un evento viene premuto viene eseguita una funzione che fornisce anche l'oggetto che descrive l'evento cliccato. Questo è la funzione che gestisce l'evento:

```
eventClick: function (info) { ... }
```

All'interno dell'oggetto *info* passato alla funzione che farà da *handler* è descritto l'evento sotto forma di oggetto JavaScript. Per accedere alle informazioni basilari dell'evento come la data, l'ora di inizio e di fine, il titolo, ecc. è possibile farlo in questo modo:

```
let event = info.event;
```

Invece per ricevere le informazioni aggiuntive, come il nome del docente che ha creato l'evento e le possibili osservazioni che ha aggiunto è possibile farlo in quest'altro modo:

```
let extra_data = info.event.extendedProps;
```

Dopo aver letto tutte le informazioni sull'evento non faccio altro che formattare i dati e scriverli all'interno di un modale tramite JQuery. Una volta scritto tutti i dati non faccio altro che aprire il modal sempre tramite JQuery. Questo è il modale che viene aperto una volta cliccato una prenotazione:

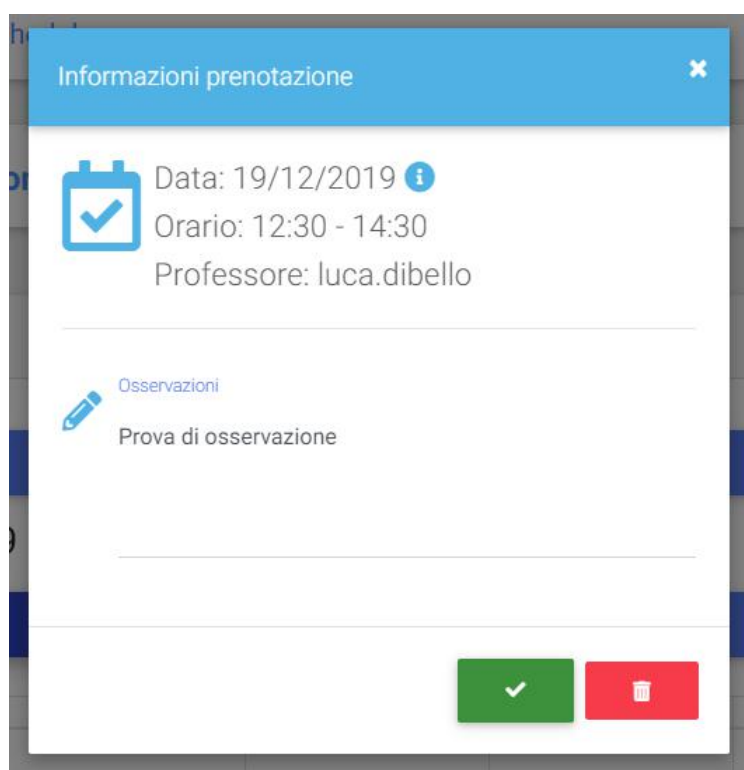


Figure 27 - Modale modifica prenotazione

Tutti i dati vengono formattati e scritti all'interno dei campi del modale. Le osservazioni sono modificabili direttamente da questa schermata. Le modifiche alle osservazioni sono applicabili cliccando la spunta verde posta sotto a destra.

Per eliminare l'evento invece si può cliccare il pulsante rosso.

Cliccando i pulsanti il sistema non fa altro che richiamare le API per la gestione degli eventi. Cliccando il pulsante verde esegue una richiesta di modifica (*update*) mentre cliccando il pulsante rosso esegue una richiesta di eliminazione (*delete*). Per ulteriori informazioni leggere il capitolo dedicato al funzionamento delle API. Le richieste alle API sono fatte tutte tramite AJAX per evitare il caricamento della pagina.

Ogni volta che si apre il modale aggiungo anche due *listener* per il click su entrambi i pulsanti (applica modifiche ed elimina). Una volta cliccati non fanno altro che leggere i dati dal modale, formattarli ed inviare la richiesta tramite le API.

4.4.3 Inserimento di prenotazioni

Per inserire una prenotazione all'interno del calendario basta cliccare su una casella del calendario. Quando viene cliccata una casella viene automaticamente richiamata una funzione che permette di avere delle informazioni sul giorno/slot orario selezionato:

```
dateClick: function (info) { ... }
```

L'oggetto info contiene tutte le informazioni necessarie per capire il giorno e l'ora selezionata. Tramite la libreria *moment.js* (<https://momentjs.com/>) sono in grado di trasformare il tutto in un oggetto:

```
var date = moment(info.dateStr);
```

Se seleziono una data ed ora oramai passata viene mostrato un errore a schermo. Questo controllo è fatto in questa maniera:

```
if (date.diff(now) < 0) {  
    // Show error  
    $.notify("La data desirata è già passata", "warn");  
}
```

Se la data risulta nel futuro invece non faccio altro che leggere tutte le informazioni necessarie e le mostro all'interno di un modale:

Figure 28 - Modale creazione prenotazione

Cliccando su annulla l'operazione di inserimento viene annullata, invece cliccando su aggiungi prenotazione il sistema legge tutti i dati inseriti nei campi del modale e li invia tramite AJAX all'API specifica per l'inserimento di nuove prenotazioni.

Una volta che le API hanno risposto viene eseguito un refresh del calendario tramite la funzione specifica (vedi capitolo 4.3.5).

4.4.4 Modifica data ed ora di un evento

Il calendario offre la possibilità di modificare la data e l'orario di un evento tramite il mouse. Per modificare la data o l'orario di una prenotazione è possibile fare un drag-and-drop dell'evento tenendo premuto il tasto destro del mouse e trascinandolo su un altro orario o giorno.

Invece si può ridurre ed allungare la durata di una prenotazione selezionando l'evento e

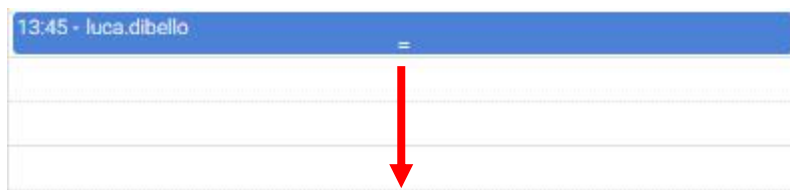


Figure 29 - Resize prenotazione

trascinando il lato in su o in giù:

Anche in questo caso la libreria fornisce due funzioni utili per rilevare queste operazioni e leggere le informazioni relative alla modifica della prenotazione.

Quando avviene un drag-and-drop di un evento viene eseguita automaticamente la seguente funzione:

```
eventDrop: function (info) { ... }
```

E invece questo metodo viene chiamato quando si esegue un *resize* della prenotazione:

```
eventResize: function (info) { ... }
```

In entrambi i casi l'oggetto *info* contiene tutte le informazioni sulla prenotazione. Questo oggetto verrà utilizzato in seguito per leggere la nuova data ed i nuovi orari di inizio e fine.

```
let date = moment(new_event.start).format("DD/MM/YYYY");
let start = moment(new_event.start).format("HH:mm");
let end = moment(new_event.end).format("HH:mm");
```

Dopo aver raccolto tutte queste informazioni viene eseguita una richiesta alle API relative alla modifica (*update*) di una prenotazione utilizzando l'API di *update* (http://example.com/api/booking/update/<id_prenotazione>).

4.4.5 Notifica errori

Se l'utente non ha i permessi per eseguire una delle operazioni disponibili (inserimento, applicazione modifiche e/o eliminazione) oppure inserisce dei dati non validi viene mostrata una notifica di errore a schermo. Questo è fatto tramite la libreria *notify.js* (<https://notifyjs.jpillora.com/>).

Quando il sistema rileva che le API hanno prodotto errori non fa altro che ciclarli ad uno ad uno tramite un *forEach* e mostrare il testo all'interno di una/più notifiche.

```
result["errors"].forEach(function (item, index, arr) {
    $.notify(item, "warn");
});
```

4.4.6 Refresh calendario

Ogni 2 minuti il calendario si aggiorna automaticamente. Questo è stato fatto per evitare di avere un calendario non aggiornato se, per esempio, si lascia la pagina aperta nel browser per molto tempo. Per aggiornare il calendario ogni 2 minuti ho utilizzato la seguente funzione:

```
/**
 * This function is used to refresh/refetch all the events of the calendar
 */
function refreshCalendar() {
    calendar.refetchEvents();
    console.log("[!] Calendar refreshed");
}

// Update every 2 mins
setInterval(refreshCalendar, 1000 * 60 * 2);
```

4.4.7 Validazione prenotazioni

Ogni dato prodotto ed inviato dalla modifica, inserimento ed eliminazione di una prenotazione viene controllato con questi 5 passaggi:

1. Controllo data nel passato
2. Controllo tempo finale maggiore di quello di inizio
3. Controllo del giorno della data di inizio e data di fine di una prenotazione
4. Controllo dell'*overlapping* delle prenotazioni

Nei sotto-capitoli sottostanti andrò a spiegare dettagliatamente come funziona ogni singolo passaggio della validazione di una prenotazione.

4.4.7.1 Step 1

Nel primo step di validazione controllo che la data della prenotazione sia nel futuro. Questo è necessario perchè un utente non deve poter inserire una prenotazione in una data oramai passata.

Questa validazione è molto semplice dato che si controlla solamente se la data della prenotazione è maggiore o meno della data corrente:

```
return $date > new DateTime();
```

4.4.7.2 Step 2

Nel secondo step vado a controllare se la data di fine prenotazione sia maggiore della data di inizio. Questo è necessario perchè un utente non può inserire un evento che finisce ancor prima di iniziare.

Anche questo controllo è molto semplice in quanto vado a controllare che la data di inizio sia minore di quella di fine:

```
return $timeStart < $timeEnd;
```

4.4.7.3 Step 3

In questo controllo invece controllo che la data di inizio prenotazione e la data di fine prenotazione appartengano allo stesso giorno. Questo controllo è necessario perchè senò un utente potrebbe prenotare l'aula per un mese intero.

Questo controllo non fa altro che controllare se le stringhe (generate dalle date di inizio e di fine) coincidano:

```
return $dateStart->format("Y-m-d") == $dateEnd->format("Y-m-d");
```

4.4.7.4 Step 4

Nella quarta validazione non faccio altro che controllare che la lunghezza (in caratteri) della possibile osservazione inserita nell'evento non superi 512 caratteri. Questo controllo è necessario in quanto all'interno del database il campo riservato per le osservazioni è di 512 caratteri.

Questo controllo non fa altro che contare i caratteri presenti nella stringa e vedere se il numero è minore o maggiore di 512:

```
return strlen($ossezazioni) < 512;
```

4.4.7.5 Step 5

Nell'ultimo step di validazione della prenotazione vado a controllare se nel lasso di tempo che occuperebbe la prenotazione sia già occupato o meno. Questo controllo è già fatto a livello font-end infatti se l'utente prova a trascinare un evento su un'altra casella già occupata (quindi con già un altro evento al suo interno) viene mostrato un errore.

Però, dato che disattivare il codice JavaScript oppure inviare richieste false alle API è molto semplice ho deciso di implementare questo controllo.

Questo è il metodo che esegue il seguente controllo:

```
public static function validateOverlapBooking(datetime $timeStart, datetime $timeEnd,
$max_overlap_events): bool {
    $overlap_events = count(BookingModel::getEventsFromRange($timeStart, $timeEnd));
    return $overlap_events >= 0 && $overlap_events <= $max_overlap_events;
}
```

Il metodo riceve ben 3 parametri: la data e l'ora di inizio della prenotazione, la data ed ora di fine della prenotazione e il numero massimo di eventi che ci possono essere nel *range* data-ora inizio - data-ora fine.

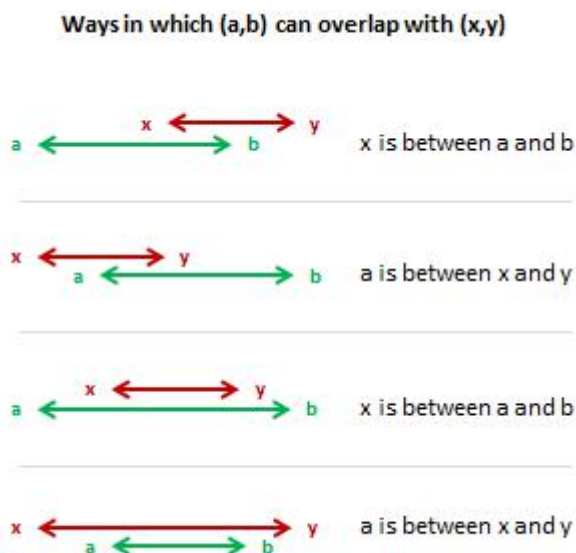
Ho dovuto inserire l'ultimo parametro in quanto, se eseguo il *resize* di un evento (quindi andando a modificare solamente l'ora di fine della prenotazione) il sistema rileverebbe che lo slot orario è già occupato.

Per ovviare a questo, quando viene eseguita una richiesta di *resize* non faccio altro che dire a questo validatore di permettere un massimo di 1 evento in quel range di tempo.

All'interno del metodo richiamo un metodo chiamato *getEventsFromRange* (presente nella classe *BookingModel*) che, come dice già il nome, permette di prendere tutti gli eventi presenti all'interno di un range. Questo è fatto tramite una lunga *query* SQL che permette di leggere gli eventi che appartengono a quel lasso di tempo. Questa è la query utilizzata per interrogare il database:

```
SELECT * FROM riservazione WHERE
DATA BETWEEN %s_dataInizio AND %s_dataFine
AND ora_inizio > NOW()
AND
(
  (ora_inizio < %d_fine AND ora_fine > %d_inizio)
  OR (ora_inizio > %d_fine AND ora_inizio < %d_inizio AND ora_fine < %d_inizio)
  OR (ora_fine < %d_inizio AND ora_fine > %d_fine AND ora_inizio < %d_fine)
  OR (ora_inizio > %d_fine AND ora_inizio < %d_inizio)
);
```

Questa query non fa altro che selezionare tutte le prenotazioni che rispettano almeno una dei seguenti 4 casi descritti in questa immagine:



Quindi seleziona tutte le prenotazioni che hanno:

Ora di inizio compresa tra l'ora di inizio e di fine di un altro evento (immagine 1):

```
(ora_inizio < %d_fine AND ora_fine > %d_inizio)
```

Ora di fine compresa tra l'ora di inizio e di fine di un altro evento (immagine 2):

(ora_inizio > %d_fine **AND** ora_inizio < %d_inizio **AND** ora_fine < %d_inizio)

Ora di inizio e fine compresa tra l'ora di inizio e di fine di un altro evento (immagine 3):

(ora_fine < %d_inizio **AND** ora_fine > %d_fine **AND** ora_inizio < %d_fine)

Ora di inizio e di fine comprende l'ora di inizio e di fine di un altro evento (immagine 4):

(ora_inizio > %d_fine **AND** ora_inizio < %d_inizio)

Per arrivare a questa soluzione ho utilizzato un post pubblicato sul seguente blog:

<https://helgesverre.com/blog/mysql-overlapping-intersecting-dates/>

Dopo aver letto dal database quante prenotazioni sono state trovate non faccio altro che controllare se il numero di prenotazioni trovare è minore del numero massimo di *overlap* concessi:

4.5 Generazione di report

La generazione del pdf di report è resa possibile tramite la libreria FPDF, la quale è stata installata molto semplicemente tramite il package manager *Composer*.

Ho creato un controller chiamato *report*, il quale si occupa di controllare se l'utente ha i permessi necessari per la generazione di report e del caricamento delle pagine (solo si dispongono dei permessi necessari). Se l'applicazione rileva che l'utente loggato ha i permessi necessari per la generazione di questi report viene mostrato un nuovo collegamento con sfondo giallo nella barra di navigazione:

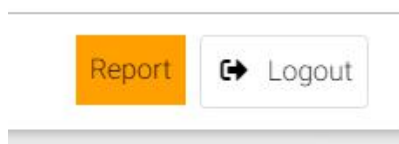


Figure 30 - Tasto report

Se un utente che non dispone dei permessi necessari per accedere a questa sezione del sito, prova ad entrarci comunque il sistema mostra una pagina di errore che notifica all'utente il problema.

Se invece l'utente ha i permessi necessari viene caricata la pagina correttamente. Questo è un screenshot della pagina:

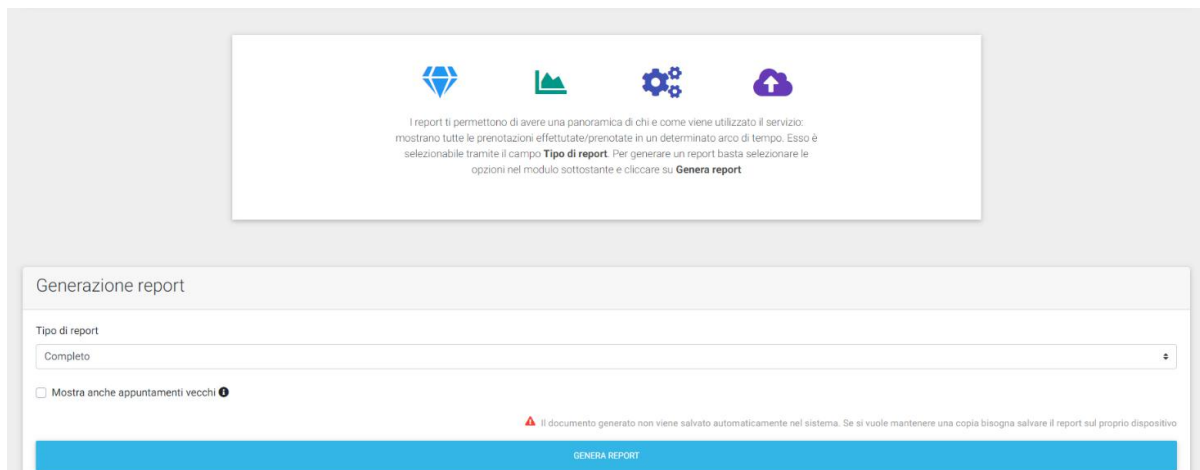


Figure 31 - Pagina generazione report

Come si può ben vedere la pagina è strutturata da due blocchi:

Il primo blocco non è altro che una spiegazione di come generare un report. Ho deciso di aggiungere questo perchè questa funzione verrà utilizzata principalmente dalla segreteria e/o la direzione e quindi ho preferito rendere il suo funzionamento il più chiaro possibile tramite una piccola spiegazione. Ecco il messaggio mostrato



Figure 32 - Spiegazione per generare un report

Il secondo blocco invece (intitolato *Generazione report*) non è altro che un modulo che permette di scegliere il tipo di report desiderato. Ecco come viene mostrato il menù all'utente:



Generazione report

Tipo di report

Completo

☐ Mostra anche appuntamenti vecchi ⓘ

⚠ Il documento generato non viene salvato automaticamente nel sistema. Se si vuole mantenere una copia bisogna salvare il report sul proprio dispositivo

GENERA REPORT

Figure 33 - Form per la generazione di report

Nel menù a tendina intitolato *Tipo di report* è possibile scegliere il tipo di report desiderato. Questi sono i vari tipi di report implementati nel sistema:

Giornaliero

- o Mostra tutte le prenotazioni del giorno in cui viene generato

Settimanale

- o Mostra tutte le prenotazioni della in cui viene generato

Mensile

- o Mostra tutte le prenotazioni del mese in cui viene generato

Annuale

- o Mostra tutte le prenotazioni dell'anno in cui viene generato

Completo

- o Mostra tutte le prenotazioni presenti nel sistema

Di default i dati mostrati in questi report sono relativi alle prenotazioni future, e quindi non vengono scritte le prenotazioni che ormai sono passate.

Se invece si abilita la spunta intitolata "*Mostra anche appuntamenti vecchi*" nel report vengono mostrati anche gli appuntamenti vecchi. Per esempio se genero un report di tipo giornaliero con abilitata la spunta verranno mostrati non solo gli appuntamenti futuri ma tutti quelli dell'intera giornata.

Una volta selezionato il tipo di report desiderato e cliccato su "*Genera report*" vengono trasmesse le impostazioni all'azione *generate* del controller (quindi <http://example.com/report/generate>). Se il tipo di record scelto non viene riconosciuto dal sistema non fa altro che riportarti al controller *report*, facendo caricare nuovamente la pagina altrimenti viene creato un oggetto di tipo *ReportGenerator* al quale vengono passate le impostazioni scelte dal form:

```
$report = new ReportGenerator($type, $all_rows_flag);
```

(Nota. *\$type* ha salvato al suo interno il tipo di report desiderato mentre *\$all_rows_flag* è un *booleano*, se esso è *true* il report mostrerà tutte le prenotazioni, altrimenti solo quelle future)

E poi viene richiamato il metodo *generate*:

```
$report->generatePDF();
```

Questo metodo esegue diverse due principali operazioni in sequenza:

1. Generazione di una query utile per la selezione dei dati corretta, in base al tipo di report selezionato (tipo + flag dati vecchi)
2. Query sul database con tramite query generata e costruzione del PDF

La generazione della query viene fatta tramite un metodo chiamato “*_generate_query*”. Questo metodo non fa altro che leggere le impostazioni inviate dal form e, a seconda delle diverse combinazioni, genera una query. Per esempio questa è la query generata per selezionare i dati necessari per la costruzione di un report settimanale normale (che mostra solo i dati vecchi):

```
SELECT utente,
DATE_FORMAT(DATA, '%d/%m/%Y') AS 'Data' ,
TIME_FORMAT(ora_inizio, '%H:%i') AS 'Ora inizio',
TIME_FORMAT(ora_fine, '%H:%i') AS 'Ora fine'
FROM riservazione
WHERE yearweek(DATA) = yearweek(CURRENT_DATE())
AND
TIMESTAMP(CONCAT(DATA, ' ', ora_inizio)) > NOW()
ORDER BY DATA, ora_inizio, ora_fine ASC
```

Come si può vedere in questa query vengono svolte diverse operazioni: nella prima parte di query i dati trovati vengono formattati in formato europeo, quindi data in formato “giorno/mese/anno” e gli orari in formato “ora/minuto”. Successivamente, nella seconda parte di query (dopo la *keyword* “*WHERE*”), avviene effettivamente fatto un “filtro” tra i dati presenti nella tabella. Il filtro viene fatto in due parti:

1. `yearweek(DATA) = yearweek(CURRENT_DATE())`
 - a. Questo primo filtro seleziona i campi con la data che appartiene alla stessa settimana della data di oggi
2. `TIMESTAMP(CONCAT(DATA, ' ', ora_inizio)) > NOW()`
 - a. Dei dati trovati in precedenza, seleziona solamente quelli nel futuro, rispetto alla data e l'ora in cui si sta generando il report. Se si genera un report con la spunta “*Mostra anche appuntamenti vecchi*” attiva il sistema non fa altro che inviare la query senza questo filtro.

Dopo aver selezionato i dati corretti essi vengono ordinati in modo ascendente a seconda della loro data, ora di inizio ed ora di fine, quindi l'appuntamento più vicino sarà alla data ed ora di generazione, più la sua posizione all'interno della tabella sarà alta.

Dopo aver generato la query, vanno recuperati i dati e creata la tabella. Per far questo ho utilizzato un pezzo di codice trovato sulla guida ufficiale di FPDF che si occupa di far questo: esegue lui il *fetch* dei dati dal database, crea l'header della tabella, esegue i calcoli per la grandezza delle celle e scrive i dati al suo interno. Questo è il link dove ho trovato il codice <http://www.fpdf.org/en/script/script14.php>

A questo codice ho fatto qualche modifica per inserire delle informazioni aggiuntive come la data di generazione, il nome dell'utente che lo ha generato ed il tipo di report generato. Ho anche dovuto aggiungere un controllo nel codice di generazione della tabella che si occupa di controllare se effettivamente il sistema ha trovato delle prenotazioni che rispettano i canoni scelti in precedenza dal modulo (quello dove si sceglie il tipo di report da generare). Se trova i dati essi vengono scritti all'interno della tabella, se invece non viene trovato nessun dato viene mostrato un messaggio che informa l'utente che non sono stati trovati dei dati che soddisfano la richiesta.

Questo è un esempio di generazione di un report di tipo annuale:

Report aula riunioni			
Data: 10/12/2019			
Generato da: luca.dibello			
Tipo: Annuale			
Utente	Data	Ora inizio	Ora fine
luca.dibello	09/12/2019	11:00	15:15
luca.dibello	12/12/2019	11:00	11:15
luca.dibello	15/10/2019	08:50	09:50
luca.dibello	17/12/2019	02:00	06:00
luca.dibello	19/12/2019	08:30	08:45

Quando ci sono due prenotazioni che appartengono a due anni diversi all'interno della tabella del report vengono separati da una linea rossa in questa maniera:

luca.dibello	10/03/2020	00:00	00:15
luca.dibello	13/12/2019	14:45	15:45

Se invece, per esempio, genero un report giornaliero ma nella giornata non è pianificata nessuna prenotazione (e quindi la query generata non riesce a trovare i dati) la tabella viene generata con al suo interno il messaggio citato in precedenza:

Report aula riunioni			
Data: 10/12/2019			
Generato da: luca.dibello			
Tipo: Giornaliero			
Utente	Data	Ora inizio	Ora fine
Non sono state trovate prenotazioni che soddisfano la richiesta.			

Questo è il pezzo di codice che ho aggiunto per mostrare il messaggio di notifica (quello in grassetto):

```
// Check if there is data or not
if(mysqli_num_rows($res) > 0){
    while ($row = mysqli_fetch_array($res))
        $this->Row($row);
}
else{
    $this->CustomTextRow
    (
        "Non sono state trovate prenotazioni che soddisfano la
        richiesta.",
        $prop["width"],
        20
    );
}
```

Il metodo *CustomTextRow* non fa altro che creare una cella di grandezza (larghezza ed altezza) custom, con al suo interno (centrato verticalmente ed orizzontalmente) un testo custom. Nel mio caso genero una cella di larghezza uguale alla tabella e di altezza 20 con al suo interno il testo che notifica l'utente che non sono state trovate le prenotazioni richieste.

Questo è il codice del metodo, il quale non è altro che una piccola rivisitazione del metodo *Row* (già presente nello snippet che ho utilizzato):

```
function CustomTextRow($text,$w,$h=5){
    $this->SetX($this->TableX);
    $ci = $this->ColorIndex;
    $fill = !empty($this->RowColors[$ci]);

    if ($fill){
        $this->SetFillColor(
            $this->RowColors[$ci][0],
            $this->RowColors[$ci][1],
            $this->RowColors[$ci][2]
        );
    }

    $this->Cell($w, $h, $text, 1, 0, 'C', $fill);
    $this->Ln();

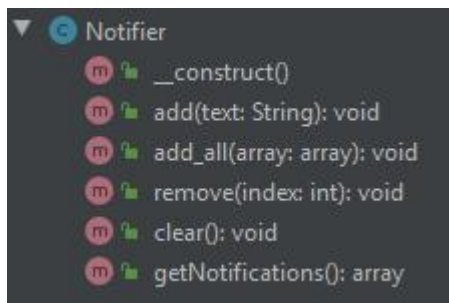
    $this->ColorIndex = 1 - $ci;
}
```

Se per qualche motivo lo script non riesce a collegarsi al database esso mostrerà un messaggio di errore, scrivendo delle informazioni utili per risolverlo (query che si ha provato ad eseguire ed informazioni sull'errore lette dal DBMS).

4.6 Gestione notifiche

La gestione delle modifiche è stata fatta tramite una piccola libreria scritta ad-hoc per questo progetto chiamata *Notifier*.

Questa classe non fa altro che gestire una lista salvata all'interno della sessione per l'utente, essa infatti fornisce delle funzioni che permettono di inserire una stringa (*add*) una lista di stringhe (*add_all*) e di pulire tutte le notifiche contenute nell'array di sessione (*clear*).



Questi sono tutte le operazioni disponibili:

Il metodo *add* ed *add_all* (come detto in precedenza) non fanno altro che aggiungere dati alla lista. Il metodo *remove* invece permette di rimuovere una notifica specifica dall'array (in questo progetto non viene utilizzato).

Un metodo fondamentale di questa classe è *getNotifications*, il quale ritorna la lista completa delle notifiche salvate all'interno di quella di sessione. Questo metodo viene utilizzato per esempio nella schermata di login per mostrare gli errori di login oppure nel pannello admin, per mostrare gli errori di validazione dei dati.

Questa classe l'ho scritta io durante il mio stage in Irlanda. Ho deciso di riutilizzarla per questo progetto dato che, una volta implementata, è molto semplice e veloce gestire i messaggi di errore.

Questo è un esempio di come si possono leggere e scrivere all'interno di una pagina web tutte le notifiche salvate:

```
<!-- Check if there are notifications -->
<?php if(count($GLOBALS["NOTIFIER"]->getNotifications()) != 0): ?>
    <!-- Write notifications one by one -->
    <?php foreach ($GLOBALS["NOTIFIER"]->getNotifications() as $notification): ?>
        <div class="alert alert-danger" role="alert">
            <?php echo $notification ?>
        </div>
    <?php endforeach; ?>
    <!-- Clear notifications -->
    <?php $GLOBALS["NOTIFIER"]->clear(); ?>
<?php endif; ?>
```

In questo esempio di codice non faccio altro che, come prima cosa, controllare se ci sono delle notifiche salvate. Se ci sono le stampo una ad una a schermo e, solo dopo averle stampate a schermo, rimuovo tutte le notifiche dalla lista. Le rimuovo per evitare di mostrarle nuovamente se l'utente ricarica la pagina.

4.7 Struttura API

Questo capitolo illustra tutte le API presenti nel sistema, spiegando la loro utilità all'interno del sistema ed il come richiamarle (spiegando anche i dati da inviare). Ho deciso di dedicare un intero capitolo a questa classe in quanto è grazie ad essa che sono possibili tutte le operazioni su utenti e prenotazioni, infatti ritengo questo controller la parte più importante del progetto: senza nessun componente del progetto andrebbe.

Questo è un piccolo schema che illustra tutte le API disponibili nel sistema:

- User
 - o Delete
 - o Add
 - o Update
 - o Promote
- Booking
 - o Delete
 - o Add
 - o Update
- Calendar
 - o Lettura di tutte le prenotazioni
 - o Lettura di tutte le prenotazioni del giorno corrente

Le API *User* e *Booking* sono delle API interne, quindi delle API accessibili solamente dal sito stesso in quanto necessitano che l'utente che richiama queste funzioni dev'essere loggato e deve avere dei permessi adeguati.

Le API relative al calendario invece possono essere utilizzate anche da sistemi esterni. Per non permettere però a chiunque di leggere le prenotazioni salvate nel database ho implementato un sistema di "password" per l'utilizzo delle API del calendario. Infatti, ogni chiamata a queste API, deve essere fornita di un token inviato sotto forma di stringa. Il sistema quando rileva la richiesta controlla il codice.

Tutte le API di questo sistema (sia quelle interne che esterne) controllano anche la bontà del dato invitato (quindi esegue una validazione). Le API interne controllano anche che l'utente sia loggato e che abbia i permessi per eseguire la seguente operazione.

Ho deciso di fare questo controllo back-end dei permessi in quanto, se fossero fatti solo lato front-end, sarebbero facilmente aggirabili disattivando JavaScript.

Le API del calendario (*Calendar*) è utilizzata dal Raspberry per leggere le prenotazioni del giorno corrente. I dati ricevuti dalle API sono sotto forma di JSON.

4.8 Esempio di API

Questo è un piccolo pezzo di codice che mostra come ho fatto a creare un API. In questo *snippet* verrà mostrata l'API per l'eliminazione di un utente del sistema:

```
public function user($action = "", $username = null)
{
    ....

    if (Auth::isAuthenticated()) {
        $GLOBALS["NOTIFIER"]->clear();

        /*
        * API url: api/user/delete/<username>
        * Permission necessari: eliminazione_utenti
        * Dati extra: nessuno
        */
        if ($action == "delete" && !is_null($username)) {
            // API creazione utenti

            // Controllo i permessi
            if (PermissionManager::getPermissions()->canEliminareUtenti()) {
                // L'utente ha i permessi per eseguire l'operazione

                // Passo al model la richiesta di eliminazione dell'utente
                $result = UserModel::delete($username);

                // Controllo se l'eliminazione dell'utente ha prodotto errori
                if (is_array($result)) {
                    // Errori trovati, li aggiungo nelle notifiche
                    $GLOBALS["NOTIFIER"]->add_all($result);
                }
            } else {
                // Se l'utente non ha i permessi aggiungo una notifica
                $GLOBALS["NOTIFIER"]->add("Non hai i permessi necessari per eliminare gli
                utenti");
            }
        }
    }
}
```

```

    }

    RedirectManager::redirect("admin/utenti");
}

}

....
}

```

Questo pattern viene ripetuto per ogni singola API del sistema.

4.8.1 API User

Nei capitoli sottostanti andrò a descrivere esaurientemente come funziona ogni singola API degli utenti. Per avere delle informazioni aggiuntive sul funzionamento di esse si può andare a leggere i commenti PHPDoc nel controller *api.php* (*src/application/controller/api.php*)

(Nota: Per usare queste API l'utente deve essere loggato nel sistema.)

4.8.1.1 Delete

Questa API viene utilizzata per eliminare un utente e tutte le sue prenotazioni dal database. Essa viene utilizzata infatti dal pannello di gestione utenti presente nel pannello di amministrazione del sistema.

Questo è link per richiamarla: http://example.com/api/user/delete/<nome_utente>

Una volta che il sistema rileverà la chiamata all'API, essa andrà a controllare se l'utente che ha eseguito questa richiesta ha effettivamente i permessi per farla. Per utilizzarla l'utente necessita del permesso di eliminazione utenti chiamato *eliminazione_utenti*.

(Nota: questa API per è richiamabile tramite qualsiasi richiesta HTML (get,post,put,delete,...) in quanto non c'è nessun tipo di controllo a livello di codice.)

4.8.1.2 Add

Questa API viene utilizzata per creare un nuovo utente all'interno del sistema. Anch'essa viene utilizzata nel pannello di gestione utenti presente nel pannello admin.

Per richiamare questa API bisogna fornire i dati dell'utente in un array associativo inviato con *POST*. Questo è un esempio di dati inviati per la creazione di un utente:

```

array (size=5)
  'nome' => string 'prova' (length=5)
  'cognome' => string 'prova' (length=5)
  'email' => string 'prova' (length=5)
  'username' => string 'prova.prova' (length=11)
  'tipo_utente' => string 'admin' (length=5)

```

Questo è il link per richiamare l'API (ed anche dove vanno inviati i dati POST):

<http://example.com/api/user/add>

Anche questa API, una volta ricevuta la richiesta, andrà a controllare se l'utente che esegue questa richiesta ha effettivamente i permessi per farlo. Per utilizzarla l'utente necessita di avere i permessi di creazione utenti chiamato *creazione_utenti*.

Se l'utente ha i permessi per eseguire l'operazione il sistema esegue un *sanitize* dei dati POST, i quali vengono successivamente passati al metodo *add* della classe *UserModel* la quale si occuperà di validazione dei dati ed inserimento di essi nel database.

4.8.1.3 Update

Questa API viene utilizzata per modificare un utente già presente all'interno del sistema. Anche questa viene utilizzata dal pannello di gestione utenti presente nel pannello admin.

Per richiamare questa API bisogna fornire tutti i dati dell'utente tramite un array associativo inviato con *POST*. Questo è un esempio di dati inviati per la modifica di un utente:

```
array (size=4)
  'username' => string 'luca.dibello' (length=12)
  'nome' => string 'Lucaa' (length=5)
  'cognome' => string 'Di Bello' (length=8)
  'email' => string 'luca.dibello' (length=12)
```

Per poter richiamare correttamente l'API questi dati vanno inviati al seguente link:

http://example.com/api/update/<nome_utente>.

Il sistema, una volta inviati i dati, controllerà se l'utente che ha eseguito la richiesta ha i permessi necessari per eseguire la seguente operazione. Per utilizzarla l'utente necessita di avere il permesso di modifica degli utenti chiamato *modifica_utenti*.

Se l'utente ha i permessi il sistema esegue un *sanitize* dei dati inviati con POST, i quali vengono successivamente passati al metodo *update* della classe *UserModel*, la quale si occuperà della validazione dei dati e della modifica dell'utente all'interno del database.

4.8.1.4 Promote

Questa API viene utilizzata per modificare i permessi di un utente già presente all'interno del sistema. Anche questa viene utilizzata dal pannello di gestione utenti presente nel pannello admin.

Per richiamare questa API bisogna fornire all'api il nome del permesso da assegnare all'utente. Questa informazione viene fornita tramite l'invio di un array associativo tramite *POST*. Questo è

```
array (size=1)
  'tipo_utente' => string 'admin' (length=5)
```

un esempio di dati inviati per la modifica dei permessi di un utente:

Per poter richiamare correttamente l'API questi dati vanno inviati al seguente link:

http://example.com/api/promote/<nome_utente>.

Il sistema, una volta inviati i dati, controllerà se l'utente che ha eseguito la richiesta ha i permessi necessari per eseguire la seguente operazione. Per utilizzarla l'utente necessita di avere il permesso di promozione degli utenti chiamato *promozione_utenti*.

Se l'utente ha i permessi il sistema esegue un *sanitize* dei dati inviati con POST, i quali vengono successivamente passati al metodo *promote* della classe *UserModel*, la quale si occuperà della validazione dei dati e della dei permessi dell'utente all'interno del database.

4.8.2 API Booking

Nei capitoli sottostanti andrò a descrivere esaurientemente come funziona ogni singola API che eseguono operazioni sulle prenotazioni. Per avere delle informazioni aggiuntive sul funzionamento di esse si può andare a leggere i commenti PHPDoc nel controller *api.php* (*src/application/controller/api.php*).

Queste API vengono utilizzate dal calendario per la modifica/creazione/eliminazione degli eventi. Per ulteriori informazioni su come vengono utilizzate/richiamate queste API dal calendario leggere il capitolo chiamato *Calendario*.

Queste API, a differenza delle API degli utenti (*User*) , ritornano sempre dei dati in JSON. In caso la richiesta viene eseguita correttamente il sistema ritorna il seguente JSON:

```
{"success": true}
```

Se invece l'operazione produce degli errori le API ritornano un JSON che contiene tutti gli errori generati. Questo è un esempio di JSON di errore:

```
{"success":false,"errors": ["Errore 1","Errore 2","Errore 3"]}
```

Ho deciso di utilizzare questo approccio in quanto queste API vengono richiamate sempre tramite JavaScript, utilizzando AJAX.

(Nota: Per usare queste API l'utente deve essere loggato nel sistema.)

4.8.2.1 Delete

Questa API viene utilizzata per eliminare una prenotazione presente all'interno del calendario. Questa viene utilizzata dalla pagina del calendario per eliminare gli evento selezionato.

Per utilizzare l'API non è necessario inviare dati aggiuntivi, infatti basta solamente eseguire una qualsiasi richiesta HTTP (post,get,delete,put,...) a questo link:

http://example.com/api/booking/delete/<booking_id>

Una volta che il sistema rileva la richiesta controlla se l'utente ha i permessi necessari per eseguirla. In questa API, a seconda dell'azione desiderata, si necessitano due tipi di permessi diversi:

Se l'utente vuole eliminare una propria prenotazione (creata da lui stesso), necessita del permesso per l'eliminazione delle prenotazioni personali chiamato *cancellazione_prenotazioni*. Se invece l'utente vuole eliminare una prenotazione che non è stata creata da lui (quindi creata da un altro utente del sistema) necessita del permesso per eliminare le prenotazioni degli altri utenti chiamato: *cancellazione_prenotazioni_altri_utenti*.

Se l'utente ha i permessi necessari per eseguire l'operazione il sistema richiama il metodo *delete* della classe *BookingModel* per eseguire l'operazione di eliminazione della prenotazione.

4.8.2.2 Add

Questa API viene utilizzata per creare una prenotazione all'interno del calendario.

Per utilizzare l'API è necessario inviare i dati che descrivono la prenotazione sotto forma di un array associativo inviato con *POST*. Questi sono i dati necessari per il funzionamento:

data

- o Data della prenotazione (formato YYYY-MM-DD)
- ora_inizio
 - o Ora di inizio della prenotazione (formato HH:mm)
- ora_fine
 - o Ora di fine della prenotazione (formato HH:mm)
- osservazioni
 - o Osservazioni aggiuntive da mostrare nella prenotazione. Se non si vuole aggiungere delle note aggiuntive basta inviare *NULL*.
- utente
 - o Username dell'utente che crea la prenotazione (es: *luca.dibello*)

I dati vanno inviati al seguente URL: <http://example.com/booking/add>

Quando il sistema rileva la richiesta esso controlla anche se l'utente che richiede la creazione di una nuova prenotazione ha effettivamente i permessi per farlo. Per eseguire l'operazione l'utente deve avere i permessi per la creazione di prenotazioni chiamato *creazione_prenotazioni*.

(Nota: È presente il permesso chiamato *creazione_prenotazioni_altri_utenti* ma non è stato implementato nel sistema)

4.8.3 API Calendar

Come detto in precedenza queste API sono *esterne*, quindi sono richiamabili anche da apparecchi esterni al sistema. Essendo esterne ho implementato un sistema di "accesso" tramite TOKEN. Ad ogni richiesta va inviato anche un token, se il token è corretto il sistema permette di eseguire l'operazione, altrimenti viene ritornato un errore. Il token è salvato all'interno del file di config (*config.php: src/application/config/config.php*).

Questo è il token di default utilizzato per l'autenticazione:

058c24b04169e44528ff2be1ac83f5dd787aa2109ad64fdcf142538f4d8617b5832e532a7c4a004398c3a3b4f12d1eac47423680fd71c02105d33c77cae12d5d

Queste API vengono utilizzate dal Raspberry per poter leggere le prenotazioni e quindi mostrarle sul monitor.

Questi sono i possibili errori che possono generare queste API:

Richiesta con metodo diverso da POST

- o {"code": 400, "message": "Wrong request"}

Token non inviato nella richiesta

- o {"code": 400, "message": "Missing API token"}

Token errato/non riconosciuto

- o {"code": 401, "message": "Wrong API token"}

4.8.3.1 Tutti i dati

Per leggere tutte le prenotazioni presenti nel sistema basta fare una richiesta POST (la quale contiene il token) al seguente URL: <http://example.com/api/calendar>

Se il token viene riconosciuto, il sistema richiama il metodo *getBookingsAfterDateTime* della classe *CalendarModel* per ritornare un array di *Booking* che descrive tutte le prenotazioni future. Esse vengono successivamente convertite in dati json e ritornati dalle API.

Questo è un esempio di risposta dell'API:

```
[{"id":18,"title":"luca.dibello","start":"2019-12-17T12:45:00+01:00","end":"2019-12-17T13:00:00+01:00","note":null,"professor":"luca.dibello"}]
```

4.8.3.2 Prenotazioni giornaliere

Per leggere tutte le prenotazioni giornaliere basta fare una richiesta POST (inviando anche il token di autenticazione) all'URL: <http://example.com/api/calendar/day>

Se il token viene riconosciuto, il sistema richiama il metodo *getBookingsAfterDateTimeDay* della classe *CalendarModel* per ritornare un array di *Booking* che descrive tutte le prenotazioni future della giornata. Esse vengono successivamente convertite in dati json e ritornati dalle API.

4.9 Schermo e Raspberry

Lo schermo è gestito completamente da un Raspberry Pi 3 Model B. Sul Raspberry ho dovuto installare il sistema operativo *Raspbian Desktop*. Ho installato la versione *Desktop* in quanto deve mostrare la pagina di un browser sul monitor.

Dopo aver installato il sistema operativo ho iniziato la creazione di una pagina web, completamente scritta in JavaScript, che legge i dati dall'API del calendario (vedi capitolo 4.8.3.2).

La pagina utilizza il framework front-end utilizzato per l'interno sito web: *Material Design Bootstrap*. Inizialmente avevo pensato di utilizzare *Bootstrap*, ma ripensandoci, ho deciso che era meglio mantenere lo stile della pagina identico allo stile del sito web.

La pagina che viene mostrata sul monitor si presenta in questa maniera:

Prenotazioni

18/12/2019

Lista prenotazioni per l'aula **A-4****

Data	Orario	Docente
18/12/2019	19:30 - 19:45	luca.dibello
18/12/2019	19:45 - 20:00	luca.dibello
18/12/2019	20:00 - 20:15	luca.dibello
18/12/2019	20:15 - 21:15	luca.dibello

Se invece, durante la giornata non è stata pianificata nessuna prenotazione viene mostrato il seguente messaggio all'interno della tabella:

Data	Orario	Docente
Non ci sono prenotazioni.		

Se le API del calendario producono un errore, esso viene mostrato sulla cima della pagina e la tabella delle prenotazioni viene nascosta:



Figure 34 - Errore mostrato sullo schermo

Questo viene fatto con ogni errore generato dalle APIs. Per ulteriori informazioni vedere il capitolo 4.8.3.

L'aggiornamento dei dati viene eseguito ogni 30 secondi. Ad ogni aggiornamento viene aggiornato il campo che mostra la data ed ora dell'ultimo aggiornamento:

Ultimo aggiornamento: **18/12/2019 19:20**

L'implementazione di questo sistema non è stata molto difficile, infatti mi ha preso solamente poche ore.

Come prima cosa carico il file di config *config.json* nell'applicazione tramite una richiesta AJAX sincrona:

```
function get_json_settings() {
    $.ajax({
        dataType: 'json',
        url: json_config_path,
        async: false,
        success: function (json) {
            console.log("[Config] Config file read");
            config = json;
        }
    });
}
```

Questo codice non fa altro che richiedere il contenuto del file JSON ed assegnarlo ad una variabile pubblica chiamata *config*. Ho deciso di eseguire una richiesta sincrona (quindi che blocca l'esecuzione del codice) in quanto il *fetch* dei dati dalle APIs necessita di avere le impostazioni definite all'interno del file di config.

Dopo aver eseguito la richiesta, e quindi dopo avere ottenuto l'oggetto *config*, eseguo un'altra richiesta AJAX alle API del calendario. Una volta aver ricevuto i dati non faccio altro che ciclarli uno ad uno e scriverli all'interno della tabella:

```
for (let i = 0; i < json.length && i < config["max_shown_booking"]; i++) {
    console.log(json[i]);

    add_row(
        json[i].start,
        json[i].start,
        json[i].end,
        json[i].title
    );
}
```

Ovviamente prima di scrivere i dati all'interno della tabella devo controllare se il JSON ritornato dalle API è un JSON che contiene tutte le prenotazioni oppure un JSON che contiene gli errori generati. Questo lo faccio semplicemente con questo *statement*:

```
if (typeof json["code"] !== 'undefined') {
    console.log("[!] Detected API error");

    // Nascondo la tabella e mostro l'errore
    $('#update-error-body').removeClass("d-none");
    $('#update-error-message').text(json["message"] + ". Codice: " + json["code"]);
    $('#booking-data').addClass("d-none");
}
else {
    console.log("[!] Booking data found");
    ...
}
```

Lo schermo viene aggiornato automaticamente ogni 30 secondi. Ogni 30 secondi non faccio altro che ricaricare il file di config, svuotare la tabella ed avviare nuovamente la funzione che fa il fetch dei dati.

4.9.1 Config

Il monitor è gestibile tramite un file di config chiamato *config.json*. Questo è un esempio di file di config:

```
{
  "api_url": "http://saminfo.ch/riservazione2019/api/calendar/day",
  "api_token": "super_secret_token",
  "max_shown_booking": 12,
}
```

Questa è la spiegazione di ogni singola opzione:

Opzione	Scopo
<i>api_url</i>	Url che il sistema andrà a contattare per leggere il JSON contenente le prenotazioni da mostrare
<i>api_token</i>	Token da inviare all'URL sovracitato per l'autenticazione
<i>max_shown_booking</i>	Il numero massimo di prenotazioni da mostrare a schermo

5 Test

5.1 Protocollo di test

Test Case:	TC-001	Nome:	Controllo struttura database
Riferimento:	REQ-001		
Descrizione:	Controllo struttura database ed utente di base		
Prerequisiti:	- Credenziali di accesso al server MySQL del sistema		
Procedura:	<ol style="list-style-type: none"> 1. Controllare che ci sia un database chiamato cptmrs 2. Controllare che all'interno del database ci siano 3 tabelle: tipo_utente, utente e riservazione 3. Controllare che la struttura delle 3 tabelle sia identica a quella descritta nel diagramma ER nel capitolo 3.2 4. Controllare che esistano nella tabella <i>tipo_utente</i> 3 record che descrivono i seguenti gruppi di utenti: <i>admin</i>, <i>avanzato</i> e <i>user</i> 5. Controllare che nella tabella <i>utente</i> ci sia almeno un utente 		
Risultati attesi:	Il database e le tabelle sono esistenti e sono strutturati correttamente. I permessi di base sono presenti e c'è almeno un utente locale.		

Test Case:	TC-002	Nome:	Login tramite utente LDAP
Riferimento:	REQ-002		
Descrizione:	Eseguire il login all'applicazione tramite un utente LDAP		
Prerequisiti:	<ul style="list-style-type: none"> - Computer connesso ad internet - Login non effettuato 		
Procedura:	<ol style="list-style-type: none"> 1. Andare nella pagina di login del sistema 2. Inserire le proprie credenziali nei campi (nome utente e password) e cliccare sul pulsante intitolato "accedi", 3. Andare nella pagina <i>info</i> tramite questo URL: http://samtinfor.ch/riservazioneale2019/info 		
Risultati attesi:	Accesso eseguito correttamente e nella pagina <i>info</i> vengono mostrate tutte le informazioni relative all'account che ha eseguito l'accesso (informazioni sull'utente ed i permessi a lui assegnati). Nel campo login_type è scritto "LDAP".		

Test Case:	TC-003	Nome:	Login tramite utente locale
Riferimento:	REQ-002		
Descrizione:	Eseguire il login all'applicazione tramite un utente locale		
Prerequisiti:	<ul style="list-style-type: none"> - Computer connesso ad internet - Login non effettuato - Credenziali di accesso di un account locale 		
Procedura:	<ol style="list-style-type: none"> 1. Andare nella pagina di login del sistema 2. Inserire le proprie credenziali nei campi (nome utente e password) e cliccare sul pulsante intitolato "accedi", 3. Andare nella pagina <i>info</i> tramite questo URL: http://samtinfor.ch/riservazioneale2019/info 		
Risultati attesi:	Accesso eseguito correttamente e nella pagina <i>info</i> vengono mostrate tutte le informazioni relative all'account che ha eseguito l'accesso (informazioni sull'utente ed i permessi a lui assegnati). Nel campo login_type è scritto "LOCAL".		

Test Case:	TC-004	Nome:	Controllo creazione utenti
Riferimento:	REQ-004		
Descrizione:	Creazione ed accesso con nuovo utente		
Prerequisiti:	- Credenziali di accesso di un account con permessi di creazione utenti		
Procedura:	<ol style="list-style-type: none"> 1. Eseguire l'accesso nel sistema ed accedere al pannello di gestione utenti nel pannello admin 2. Creazione di un nuovo utente (N.B inserire una mail reale a cui si ha accesso) 3. Login al sistema utilizzando le credenziali ricevute per mail 4. Password dell'utente cambiata con successo 5. Login al sistema utilizzando la password cambiata 6. Creazione di un evento all'interno del calendario 		
Risultati attesi:	Email con le credenziali di accesso ricevuta correttamente. Al primo login la password di accesso viene cambiata correttamente. Dopo essersi loggati con la nuova password si riesce a creare correttamente un nuovo evento all'interno del calendario.		

CPT Meeting Room Scheduler

Test Case:	TC-005	Nome:	Controlli sugli eventi del calendario
Riferimento:	REQ-005		
Descrizione:	Controllo approfondito di tutte le funzioni disponibili nel calendario		
Prerequisiti:	- Credenziali di accesso di un account i permessi di visualizzazione, modifica ed eliminazione di eventi all'interno del calendario		
Procedura:	<ol style="list-style-type: none"> 1. Eseguire l'accesso nel sistema ed accedere alla pagina del calendario 2. Provare a creare un evento nella data 01.01.2000 3. Provare a creare un evento nel giorno corrente ma ad un orario passato 4. Provare a creare un evento nel giorno corrente ma ad un orario futuro 5. Provare a creare un evento nello slot orario successivo all'evento creato in precedenza 6. Provare a spostare (drag-and-drop), ridimensionare e modificare le osservazioni 7. Eliminare l'ultimo evento creato 		
Risultati attesi:	Nel punto 2. e nel punto 3. viene mostrato un errore a schermo. Le altre operazioni vengono eseguite senza problemi		

Test Case:	TC-006	Nome:	Controllo funzionamento schermo con RaspBerry
Riferimento:	REQ-006		
Descrizione:	Controllo del corretto funzionamento dello schermo e RaspBerry		
Prerequisiti:	- Credenziali di accesso di un account i permessi di visualizzazione, modifica ed eliminazione di eventi all'interno del calendario		
Procedura:	<ol style="list-style-type: none"> 1. Eseguire l'accesso nel sistema ed accedere alla pagina del calendario 2. Creazione di 3 prenotazioni in giorni ed orari diversi tra loro 3. Aspettare 1 minuto di tempo 4. Controllare lo schermo 5. Eliminare i 3 eventi 6. Aspettare 1 minuto di tempo 7. Controllare lo schermo 		
Risultati attesi:	Dopo aver creato i 3 eventi ed aver aspettato 1 minuto essi verranno mostrati sullo schermo gestito dal RaspBerry. Una volta eliminati e dopo aver aspettato un altro minuto esse scompariranno dallo schermo		

Test Case:	TC-007	Nome:	Controllo generazione di report
Riferimento:	REQ-007		
Descrizione:	Controllo della corretta generazione dei report		
Prerequisiti:	<ul style="list-style-type: none"> - Credenziali di accesso di un account i permessi di generazione di report - Prenotazioni presenti nel sistema 		
Procedura:	<ol style="list-style-type: none"> 1. Entrare nella pagina dedicata alla generazione dei report 2. Provare a generare ogni tipo di report disponibile: <ol style="list-style-type: none"> a. Giornaliero b. Settimanale c. Mensile d. Annuale e. Completo 		
Risultati attesi:	I report mostrano i dati correttamente ed i dati scritti nella tabella sono ordinati dal più recente al più vecchio.		

5.2 Risultati test

Nome test	Stato
Controllo struttura database	Passato
Login tramite utente LDAP	Passato
Login tramite utente locale	Passato
Controllo creazione utenti	Passato
Controlli sugli eventi del calendario	Passato
Controllo funzionamento schermo con RaspBerry	Passato
Controllo generazione di report	Passato

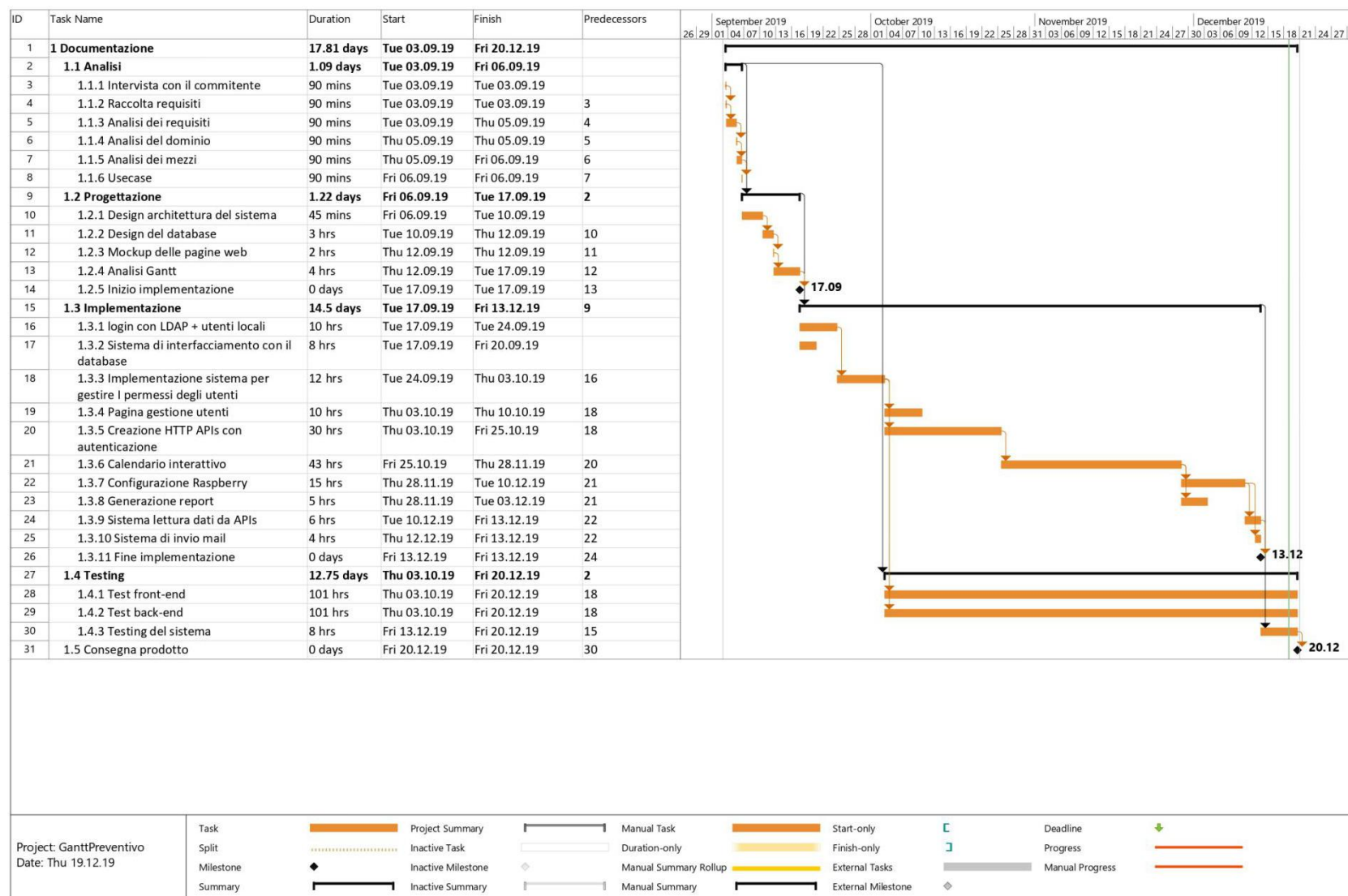
Tutti i test effettuati sui requisiti stilati durante l'inizio del progetto sono stati passati correttamente.

5.3 Mancanze/limitazioni conosciute

Il sistema supporta la gestione dei permessi ma non offre una pagina all'interno del pannello admin per modificarli in modo interattivo. Ora per aggiungere/modificare un permesso ad un utente bisogna modificarlo all'interno della banca dati manualmente.

Un'altra mancanza di questo progetto è il fatto che manca una pagina per la gestione delle prenotazioni all'interno del pannello admin. Un amministratore può comunque eliminare, spostare e modificare le prenotazioni degli utenti ma esse possono essere molto difficili da trovare all'interno del calendario interattivo.

6 Consuntivo



Come si può vedere bene dal grafico, i lavori sono stati svolti in un ordine leggermente differente da quello pianificato inizialmente.

La parte relativa all'analisi ed alla pianificazione sono rimaste quasi identiche mentre la parte relativa alla implementazione ed quella relativa al *testing* sono state stravolte completamente.

Ci sono diverse attività, come l'attività riguardante lo sviluppo del sistema di login e quella riguardanti la creazione di un sistema di interfacciamento con il database, che sono state eseguite in parallelo. Questo perché, mentre implementavo una parte di codice per una determinata funzionalità dovevo modificare del codice scritto in precedenza in quanto sbagliato o non funzionante.

Questa cosa è capitata molteplici volte durante lo sviluppo in quanto tutto il sistema è collegato, quindi è molto semplice fare questo tipo di errore.

Le attività relative al *testing* del sistema non sono state fatte solamente dopo l'implementazione (come pensato inizialmente) ma sono presenti per la maggior parte dell'implementazione. Questo perché, dopo aver implementato una nuova funzionalità nel sito web ho dovuto testarla per controllare se effettivamente funzionava correttamente.

Mi sono reso conto che il diagramma di Gantt preventivo era molto impreciso. Questo è probabilmente dovuto dal fatto che nei progetti scorsi ho sempre fatto fare la pianificazione da un altro componente del gruppo.

7 Conclusioni

La mia soluzione permetterà di rendere più semplice la prenotazione e la gestione dell'aula al quarto piano. Ritengo questo un traguardo molto importante in quanto è da anni che questo progetto viene proposto ma risulta non funzionante o poco funzionante (tanti *bug*).

Dico "semplificherà la prenotazione dell'aula" in quanto con questo sistema i docenti possono prenotare l'aula direttamente dal loro *smartphone*, *tablet* o *computer* dovunque loro siano. Grazie al calendario interattivo l'aggiunta, la modifica e la visualizzazione delle prenotazioni risulta semplice e fluida.

Invece quando dico "semplificherà la gestione" intendo la gestione da parte della segreteria. Con lo strumento per la generazione dei report ed i 5 tipi di report disponibili la direzione sarà in grado di sapere con facilità chi ha utilizzato l'aula e quando è stata utilizzata.

Rimpiazzando il foglio di carta appeso sulla porta si potranno risolvere possibili problemi di perdita di informazioni (foglio perso e/o rovinato) e illeggibilità delle informazioni (calligrafia non leggibile).

Grazie al monitor attaccato nel corridoio sarà molto più semplice sia per gli allievi, che per genitori e docenti capire quando sarà occupata o da chi è occupata in quel momento.

7.1 Sviluppi futuri

Il calendario, se utilizzato su dispositivi mobili con schermo *touch* (es: *smartphone* o *tablet*), non offre tutte le funzioni offerte nella versione *desktop* (computer). Da dispositivi mobili è possibile sempre creare, modificare e spostare gli eventi però non è possibile l'estensione di un evento su più ore.

Uno sviluppo futuro utile sarebbe (come citato nel capitolo 4.3) sarebbe la creazione di una pagina all'interno del pannello admin che permette di gestire in modo interattivo i permessi ed i gruppi di permessi. Il sistema che controlla i permessi degli utenti è già implementato però se per esempio bisogna modificare i permessi del gruppo *admin* bisogna modificare i dati direttamente dalla banca dati.

Un'altro possibile sviluppo futuro un po' più superfluo sarebbe la generazione di report più di bell'aspetto (*header*, *footer*, immagini, ...).

7.2 Considerazioni personali

Con questo progetto ho imparato come creare e gestire un calendario completamente utilizzando funzioni front-end, quindi tramite JavaScript ed AJAX. Ho imparato come gestire i permessi degli utenti sia a livello di banca dati sia a livello di applicativo (controlli, assegnazione permessi, lettura permessi).

Devo ammettere che non è stata semplice l'implementazione di questo sistema in quanto offre diverse funzionalità (calendario interattivo, generazione report, gestione utenti, ...).

8 Bibliografia

8.1 Sitografia

1. <https://dev.mysql.com/doc/>, *MySQL :: MySQL Documentation*, Visitato molteplici volte durante l'implementazione
2. <https://mdbootstrap.com/docs/jquery/>, *Material Design Bootstrap UI Kit*, Visitato molteplici volte durante l'implementazione
3. <https://stackoverflow.com/>, *StackOverflow*, Visitato molteplici volte durante l'implementazione
4. <http://www.fpdf.org/>, *FPDF*, 03-12-2019
5. <https://www.w3schools.com/>, *W3School Online web tutorial*, Visitato molteplici volte durante l'implementazione
6. <https://www.php.net/>, *PHP: Documentation*, Visitato molteplici volte durante l'implementazione

9 Allegati

Elenco degli allegati, esempio:

Diari di lavoro
Codice sorgente
Qdc
Prodotto