Gestione richieste congedi docenti

[1 Introduzione 4](#_Toc34205386)

[1.1 Informazioni sul progetto 4](#_Toc34205387)

[1.2 Abstract 4](#_Toc34205388)

[1.3 Scopo 4](#_Toc34205389)

[2 Analisi 5](#_Toc34205390)

[2.1 Analisi del dominio 5](#_Toc34205391)

[2.2 Analisi e specifica dei requisiti 5](#_Toc34205392)

[2.3 Use case 9](#_Toc34205393)

[2.3.1 Utenti di dominio 9](#_Toc34205394)

[2.3.2 Utenti locali 10](#_Toc34205395)

[2.4 Pianificazione 11](#_Toc34205396)

[2.4.1 Analisi 12](#_Toc34205397)

[2.4.2 Progettazione 12](#_Toc34205398)

[2.4.3 Implementazione 12](#_Toc34205399)

[2.4.4 Testing 13](#_Toc34205400)

[2.4.5 Consegna 13](#_Toc34205401)

[2.5 Analisi dei mezzi 13](#_Toc34205402)

[2.5.1 Software 13](#_Toc34205403)

[2.5.2 Hardware 13](#_Toc34205404)

[3 Progettazione 14](#_Toc34205405)

[3.1 Schema di rete concettuale 14](#_Toc34205406)

[3.2 Design dei dati e database 14](#_Toc34205407)

[3.2.1 Schema ER 15](#_Toc34205408)

[3.2.2 Descrizioni delle tabelle 15](#_Toc34205409)

[3.2.3 Schema logico 18](#_Toc34205410)

[3.3 Design delle interfacce 18](#_Toc34205411)

[3.3.1 Pagina di accesso 18](#_Toc34205412)

[3.3.2 Pagina principale 19](#_Toc34205413)

[3.3.3 Pagina di un recipiente 20](#_Toc34205414)

[3.3.4 Pagina storico 20](#_Toc34205415)

[3.3.5 Pagina amministrazione utenti 21](#_Toc34205416)

[3.3.6 Pagina amministrazione congedi 22](#_Toc34205417)

[3.4 Design dell’architettura del sistema 22](#_Toc34205418)

[4 Implementazione 22](#_Toc34205419)

[4.1 Gestione versioni 22](#_Toc34205420)

[4.2 Gestore di pacchetti 22](#_Toc34205421)

[4.3 Database 23](#_Toc34205422)

[4.4 Applicativo Web 24](#_Toc34205423)

[4.4.1 Struttura 24](#_Toc34205424)

[4.4.2 Configurazione 24](#_Toc34205425)

[4.4.3 Autenticazione 25](#_Toc34205426)

[4.4.3.1 Autenticazione banca dati locale 25](#_Toc34205427)

[4.4.3.2 Interfaccia LDAP 26](#_Toc34205428)

[4.4.3.3 Gestione delle sessioni e percorsi 27](#_Toc34205429)

[4.4.4 Interfaccia invio email 30](#_Toc34205430)

[4.4.5 Validazione campi 30](#_Toc34205431)

[4.4.6 Risorse locali 31](#_Toc34205432)

[4.4.7 Sicurezza 32](#_Toc34205433)

[4.4.7.1 Password locali 32](#_Toc34205434)

[4.4.7.2 Recupero password 32](#_Toc34205435)

[5 Test 34](#_Toc34205436)

[5.1 Protocollo di test 34](#_Toc34205437)

[5.2 Risultati test 34](#_Toc34205438)

[5.3 Mancanze/limitazioni conosciute 34](#_Toc34205439)

[6 Consuntivo 35](#_Toc34205440)

[7 Conclusioni 35](#_Toc34205441)

[7.1 Sviluppi futuri 35](#_Toc34205442)

[7.2 Considerazioni personali 35](#_Toc34205443)

[8 Bibliografia 35](#_Toc34205444)

[8.1 Sitografia 35](#_Toc34205445)

[9 Glossario 35](#_Toc34205446)

[10 Allegati 35](#_Toc34205447)

# Introduzione

## Informazioni sul progetto

Allievi coinvolti nel progetto: Filippo Finke

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

Docenti responsabili: Fabrizio Valsangiacomo

Data inizio: 23.01.2020

Data consegna: 06.04.2020

## Abstract

*The aim of the project "Gestione richieste congedi docenti" is to transfer the management system for teachers' leave requests to a computer system. The aim is therefore to make it easier for teachers to write their leave requests, by integrating a system of access to the web application through the school domain accounts using the LDAP protocol and to simplify the work regarding the management of these requests by the school administration (secretariat, deputy directors and management). This application will then be used by teachers if they have to plan a leave of absence. The application also makes the drafting and management of leave much faster than paper, thanks to the digital system will be possible to perform detailed research, in-depth checks and additional features that would take a long time on paper. The leave will pass through different containers inside the web application where they will be reviewed, approved, modified or refused by the school administration. Thanks to this system you will also be able to keep track of your requested leave and take their status into account. There will also be an automatic system for the creation of the paper representation of leave through the use of PDF .*

## Scopo

Lo scopo del progetto “Gestione richieste congedi docenti” è quello di trasferire il sistema di gestione delle richieste di congedo dei docenti su un sistema informatico. Lo scopo è quindi di rendere più semplice la stesura dei congedi da parte dei docenti, integrando un sistema di accesso all’applicativo web attraverso gli account del dominio scolastico sfruttando il protocollo LDAP e di semplificare il lavoro per quanto riguarda la gestione di essi da parte dalla amministrazione scolastica (segreteria, vice direttori e direzione). Questo applicativo verrà quindi utilizzato dai docenti qualora dovessero pianificare un congedo. L’applicativo inoltre rende la stesura e la gestione dei congedi molto più veloce rispetto alla carta, grazie al sistema digitale sarà possibile eseguire ricerche dettagliate, controlli approfonditi e ulteriori funzionalità che su carta prederebbero molto tempo. I congedi attraverseranno diversi contenitori presenti all’interno dell’applicativo web nel quale verranno revisionati, approvati, modificati o rifiutati da parte dell’amministrazione scolastica. Grazie a questo sistema si potrà tenere traccia anche dei propri congedi richiesti e tenerne conto dello stato. Sarà inoltre presente un sistema automatico per la creazione della rappresentazione cartacea dei congedi attraverso l’utilizzo di PDF.

# Analisi

## Analisi del dominio

È stato richiesto lo sviluppo di un gestionale web per la richiesta di congedo dei docenti del Centro Professionale di Trevano. Il prodotto dovrà essere un applicativo web, accessibile attraverso la rete utilizzando qualsiasi browser. Gli utenti che accederanno a questo applicativo saranno principalmente i docenti che andranno ad inserire le proprie richieste di concedo. Inoltre saranno disponibili ulteriori permessi per la gestione di congedi. Le varie richieste da parte dei docenti verranno gestite attraverso dei contenitori virtuali i quali saranno accessibili solamente a determinati gruppi di utenti. Vi sarà un contenitore dedicato alla segreteria la quale riceverà i congedi inviati dai docenti ed avrà il compito di verificare la validità dei dati immessi e di approvare il congedo per il passaggio al recipiente successivo. Il recipiente successivo sarà accessibile da vice direzione e direzione i quali avranno la parola definitiva sullo stato del congedo, ovvero se è stato accettato oppure no. I permessi saranno a cascata quindi chi farà parte del gruppo di direzione potrà vedere tutti i recipienti sottostanti. È inoltre richiesta una pagina accessibile solamente a chi è amministratore locale dell’applicativo e che quindi si occuperà della gestione di esso (aggiunta di titoli di congedi, descrizioni, gestione amministratori locali, etc.). Il sistema inoltre utilizzerà un sistema di posta elettronica per tenere informati sia i docenti che l’amministrazione scolastica sullo stato dei vari congedi. Quando un congedo sarà approvato definitivamente dalla direzione scolastica verrà generato un PDF di esso che verrà poi inviato per email ai diretti interessati. All’interno dell’applicativo sarà inoltre presente uno storico dei congedi registrati all’interno del sistema in modo di permettere all’amministrazione di eseguire ricerche e controlli. L’accesso all’applicativo verrà gestito attraverso l’utilizzo di due diversi sistemi, il primo sistema di autenticazione che verrà controllato sarà l’accesso locale all’applicativo per permettere ai gestori del prodotto di ricevere ulteriori permessi. Il secondo sistema che verrà utilizzato più spesso sarà l’accesso attraverso le stesse credenziali dei profili della rete scolastica attraverso l’interrogazione dei server LDAP.

DA\_REVISIONARE

## Analisi e specifica dei requisiti

È richiesto da parte del committente lo sviluppo di un applicativo che sia accessibile tramite rete e che sia web. Il prodotto deve potersi interfacciare con i server LDAP della scuola in modo da permettere l’accesso da parte dei docenti utilizzando le credenziali del dominio. È inoltre richiesto che vengano gestiti a parte gli utenti amministratore che si occuperanno della gestione del software. Attraverso l’autenticazione LDAP l’applicativo dovrà riconoscere il livello di permesso dell’utente e se farà parte di uno dei seguenti gruppi:

* Docente
* Segreteria
* Vice direttori
* Direttori o Direttrici

Attraverso questi gruppi ricavati dai server scolastici sarà possibile dunque distinguere la tipologia di utente e di come verranno mostrate le pagine stesse. Per eseguire l’autenticazione è dunque richiesta una pagina dedicata, la quale avrà anche la funzione di recupero password per gli utenti locali. Per distinguere tra utenti di gestione dell’applicativo e utenti del dominio del CPT verrà utilizzato il nominativo, che per gli utenti locali sarà il proprio indirizzo di posta elettronica mentre per gli utenti del CPT sarà “nome.cognome”. Il sistema richiede spesso l’utilizzo di email quindi è richiesto di implementare una parte del programma che si occupi solamente della gestione dell’invio di posta elettronica. Sarà presente una struttura a recipienti nel quale verranno salvati i vari congedi, questa struttura varierà in modo dinamico in base ai permessi citati in precedenza. Una volta eseguito l’accesso gli utenti avranno la possibilità di compilare il formulario dei congedi ed inviarlo al primo recipiente, ovvero la segreteria. Per tenere traccia dell’andamento dei congedi scolastici è inoltre richiesta una sezione separata nel quale mostrare tutti i congedi registrati nel sistema, attraverso questa pagina sarà anche possibile eseguire delle ricerche mirate. Per la gestione dell’applicativo invece è richiesta una sezione riservata solamente agli utenti amministratori locali che avranno la possibilità di aggiungerne altri oppure eliminarli. All’aggiunta di un amministratore locale verrà inviata un email all’indirizzo specificato contenente una password generata in modo casuale che verrà poi sostituita al primo accesso. Una sezione aggiuntiva nel pannello di amministrazione è la possibilità di gestire le cause e/o motivazioni dei vari congedi.

DA\_REVISIONARE

|  |  |
| --- | --- |
| **ID: REQ-000** | |
| **Nome** | Piattaforma dell’applicativo |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | Si necessita di un applicativo web. |

|  |  |
| --- | --- |
| **ID: REQ-001** | |
| **Nome** | Interfaccia con server LDAP scolastico |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | Si necessita di interfacciarsi con i server LDAP scolastici per eseguire il login degli utenti normali. |
| **Sotto requisiti** | |
| **001** | Possibilità di accedere con username formato “nome.cognome” e password LDAP. |
| **002** | Si necessita di ricavare i permessi da LDAP (amministrazione, etc.). |

|  |  |
| --- | --- |
| **ID: REQ-002** | |
| **Nome** | Interfaccia con server MySQL locale |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | Si necessita di interfacciarsi con un server MySQL per autenticare gli utenti locali di gestione del software. |
| **Sotto requisiti** | |
| **001** | Possibilità di accedere con email come username. |

|  |  |
| --- | --- |
| **ID: REQ-003** | |
| **Nome** | Interfaccia per invio posta elettronica |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | Si necessita di una interfaccia che permetta di inviare semplicemente messaggi di posta elettronica. |

|  |  |
| --- | --- |
| **ID: REQ-004** | |
| **Nome** | Pagina di accesso |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | Si necessita di una pagina specifica per l’accesso all’applicativo web. / Dipende dal requisito REQ-001, REQ-002, REQ-003 |
| **Sotto requisiti** | |
| **001** | Si necessita una maschera di login |
| **002** | Si necessita di poter distinguere la tipologia di accesso (LDAP o locale). |
| **003** | Finestra a comparsa al primo login per gli utenti locali. |
| **004** | Possibilità di recuperare la password, solamente per gli utenti locali, tramite email. |

|  |  |
| --- | --- |
| **ID: REQ-005** | |
| **Nome** | Pagina di un recipiente |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | Si richiede l’utilizzo di “recipienti” virtuali per organizzare i congedi, verrà utilizzata una sola pagina generata in modo dinamico per ogni contenitore. / Dipende dal requisito REQ-001, REQ-002, REQ-003 |
| **Sotto requisiti** | |
| **001** | Sarà accessibile solamente a chi ne ha i permessi (segreteria, vice direttori o direzione). |
| **002** | Possibilità di vedere i congedi al suo interno. |
| **003** | Possibilità di modificare dati dei congedi. |
| **004** | Possibilità di mandare i congedi ad un recipiente successivo. |

|  |  |
| --- | --- |
| **ID: REQ-006** | |
| **Nome** | Pagina principale |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | Si richiede una pagina principale nel quale i docenti del CPT possano creare ed inviare i congedi. Dipende dal requisito REQ-001, REQ-002, REQ-003 |
| **Sotto requisiti** | |
| **001** | Possibilità di selezionare le cause del congedo. |
| **002** | Possibilità di inviare un congedo al recipiente successivo. |
| **003** | In caso si abbiano i permessi mostrare tutti i recipienti. |
| **004** | In caso si abbiano i permessi mostrare la pagina di storico. |

|  |  |
| --- | --- |
| **ID: REQ-007** | |
| **Nome** | Pagina storico |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | Si richiede una pagina nel quale venga mostrato lo storico dei congedi salvati nel sistema. Dipende dal requisito REQ-001, REQ-002, REQ-003 |
| **Sotto requisiti** | |
| **001** | Possibilità di accedere solamente se si ha determinati permessi. |
| **002** | Possibilità di eseguire ricerche. |
| **003** | I congedi dovranno essere classificati per anno. |

|  |  |
| --- | --- |
| **ID: REQ-008** | |
| **Nome** | Pagina di amministrazione utenti |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | Si necessita di una pagina per la gestione degli utenti locali dell’applicativo. Dipende dal requisito REQ-001, REQ-002, REQ-003 |
| **Sotto requisiti** | |
| **001** | Accessibile unicamente dagli utenti amministratori locali. |
| **002** | Possibilità di vedere tutti gli utenti locali disponibili. |
| **003** | Possibilità di aggiungere utenti locali con nome, cognome, email come identificativo. |
| **004** | Possibilità di eliminare utenti locali (non si può eliminare se stessi) |

|  |  |
| --- | --- |
| **ID: REQ-009** | |
| **Nome** | Pagina di amministrazione congedi |
| **Priorità** | 2 |
| **Versione** | 1.0 |
| **Note** | Si necessita di una pagina per la gestione dei titoli delle motivazioni dei congedi. |
| **Sotto requisiti** | |
| **001** | Possibilità di aggiungere tipologie di congedi. |

## Use case

### Utenti di dominio

In questo schema gli agenti hanno già eseguito l’accesso attraverso il server LDAP della scuola. Sono presenti quattro diversi livelli di permessi all’interno dell’applicativo web che sono rappresentati dagli agenti. Gli utenti comuni sono considerati “Docente”, hanno la possibilità di accedere ad una pagina principale del sito web e di visionare il proprio contenitore personale. Inoltre possono anche compilare ed inviare congedi al livello successivo. Un'altra tipologia di utente è la “Segreteria” essa ha la possibilità di fare tutto ciò che un “Docente” può fare, ma in aggiunta può vedere tutti i congedi in attesa di revisione, che una volta revisionati possono essere mandati al livello successivo. Il livello successivo è la vice direzione e direzione, a questo livello di permessi è possibile eseguire tutto ciò che la “Segreteria” può fare, in più è possibile approvare o rifiutare i congedi in attesa. In più del “Docente” tutti gli altri agenti hanno la possibilità di accedere allo storico dei congedi nel quale è possibile eseguire delle ricerche.

DA\_REVISIONARE



Figura 1 Schema caso d'uso utenti di dominio.

### Utenti locali

In questo schema l’agente è uno solo, lo stato iniziale è che ha già eseguito l’accesso all’applicativo web. L’amministratore può gestire l’intero applicativo. Una volta entrato ha la possibilità di recarsi alla gestione degli utenti, nella quale può aggiungere, modificare ed eliminare utenti locali. Inoltre ha la possibilità di recarsi in un'altra pagina dedicata alla gestione delle cause/motivazioni dei congedi, attraverso questa pagina può aggiungerne, modificarne oppure eliminare.

DA\_REVISIONARE



Figura 2 Schema caso d'uso utenti locali.

## Pianificazione



Figura 3 Diagramma di Gantt preventivo.

### Analisi

Ho suddiviso la fase di analisi in cinque attività̀ principali. Questa è la fase più̀ corta rispetto alle altre fasi, come anche descritto nel quaderno dei compiti. Durante questa fase mi sono occupato di capire di cosa tratta il progetto e quali sono le richieste da parte del committente. DA\_REVISIONARE



Figura 4 Diagramma di Gantt, Analisi.

### Progettazione

La progettazione è la seconda fase del progetto “Gestione richiesta congedi docenti” composta da cinque attività. All’interno di questa fase di progettazione è incluso la creazione del diagramma di Gantt da seguire durante la durata del progetto, il design dell’architettura del sistema, il design dei dati e dei database, design delle interfacce ed in fine il design procedurale. Considero questa fase molto importante in quanto le fasi successive saranno basate su di essa. DA\_REVISIONARE



Figura 5 Diagramma di Gantt, Progettazione.

### Implementazione

La fase di implementazione è la più lunga all’interno del progetto. Questa è la fase più lunga del progetto, consiste nella vera scrittura del codice che andrà a comporre il prodotto finale. Questa fase è strettamente collegata all’analisi e alla progettazione in quanto si basa su di esse per lo sviluppo. All’interno di questa sezione vi è la configurazione per l’utilizzo di REST, lo sviluppo della banca dati, lo sviluppo di classi per agevolare l’autenticazione attraverso LDAP e classi per la gestione dell’invio di posta elettronica. Inoltre è presente una pietra miliare per indicare lo sviluppo del backend. Vi è una sotto fase che consiste nello sviluppo delle varie interfacce grafiche, alla fine dello sviluppo di esse vi è un'altra pietra miliare che indica il termine del frontend. DA\_REVISIONARE



Figura 6 Diagramma di Gantt, Implementazione.

### Testing

Un’altra fase molto importante è il testing, in questo caso viene data priorità di test delle classi di autenticazione LDAP e di invio di posta elettronica. È presente inoltre un'altra attività dedicata al test generale del prodotto una volta terminato frontend e backend. DA\_REVISIONARE



Figura 7 Diagramma di Gantt, Testing.

### Consegna

L’ultima fase consiste nella consegna del prodotto, è presente una sola attività che riguarda la documentazione. Questo perché verrà aggiornata durante il corso di tutto il progetto. DA\_REVISIONARE



Figura 8 Diagramma di Gantt, Consegna.

## Analisi dei mezzi

### Software

I software utilizzati per la realizzazione del progetto sono:

* Google Chrome 76.0
* Microsoft Word 2016
* Microsoft Project 2019
* Microsoft VS Code 1.37.1
* VMware Fusion 11.0
* MySQL 8.0.13
* PHP 7.3.5
* Draw.io (<https://draw.io>)
* HighlightCode (<https://highlight.hohli.com>)

Librerie utilizzate:

* jQuery 3.4.1 (https://jquery.com/)
* Bootstrap 4.3.1 (<https://getbootstrap.com/>)
* php-rest (<https://github.com/filippofinke/php-rest>)
* FPDF (<http://www.fpdf.org/>)

Template utilizzati:

* desk-app (<https://github.com/dropways/deskapp>)

Gestore librerie utilizzato:

* Composer 1.7.3 (<https://getcomposer.org/download/>)

DA\_REVISIONARE

### Hardware

Il progetto è stato sviluppato su un MacBook Pro 2018.

Le specifiche hardware sono:

* 8 GB di RAM
* Intel Core I5 4 core

Il progetto potrà essere messo in produzione su una qualsiasi macchina con più di:

* 512MB di RAM
* 2GB di disco

DA\_REVISIONARE

# Progettazione

## Schema di rete concettuale

Un computer che vuole collegarsi all’applicativo web deve essere collegato ad una rete nella quale sia presente anche il web server che si occupa di servire il sito. Questa rete può locale oppure pubblica. Quando i due dispositivi saranno connessi nella stessa rete basterà conoscere l’indirizzo IP del web server per potersi collegare. Il computer per collegarsi deve possedere un software in grado di interpretare il protocollo di comunicazione, quindi un semplice browser. Inoltre quando l’accesso viene eseguito attraverso le credenziali del dominio LDAP il Web Server attraverso l’applicativo comunicherà con il server scolastico per la convalida dei dati stessi. DA\_REVISIONARE



Figura 9 Schema di rete basilare.

## Design dei dati e database

I dati dell’applicativo verranno salvati all’interno di un database MySQL. Verrà anche utilizzato un server LDAP, già presente nella rete scolastica, utilizzato per l’autenticazione con gli account di dominio.

DA\_REVISIONARE

### Schema ER

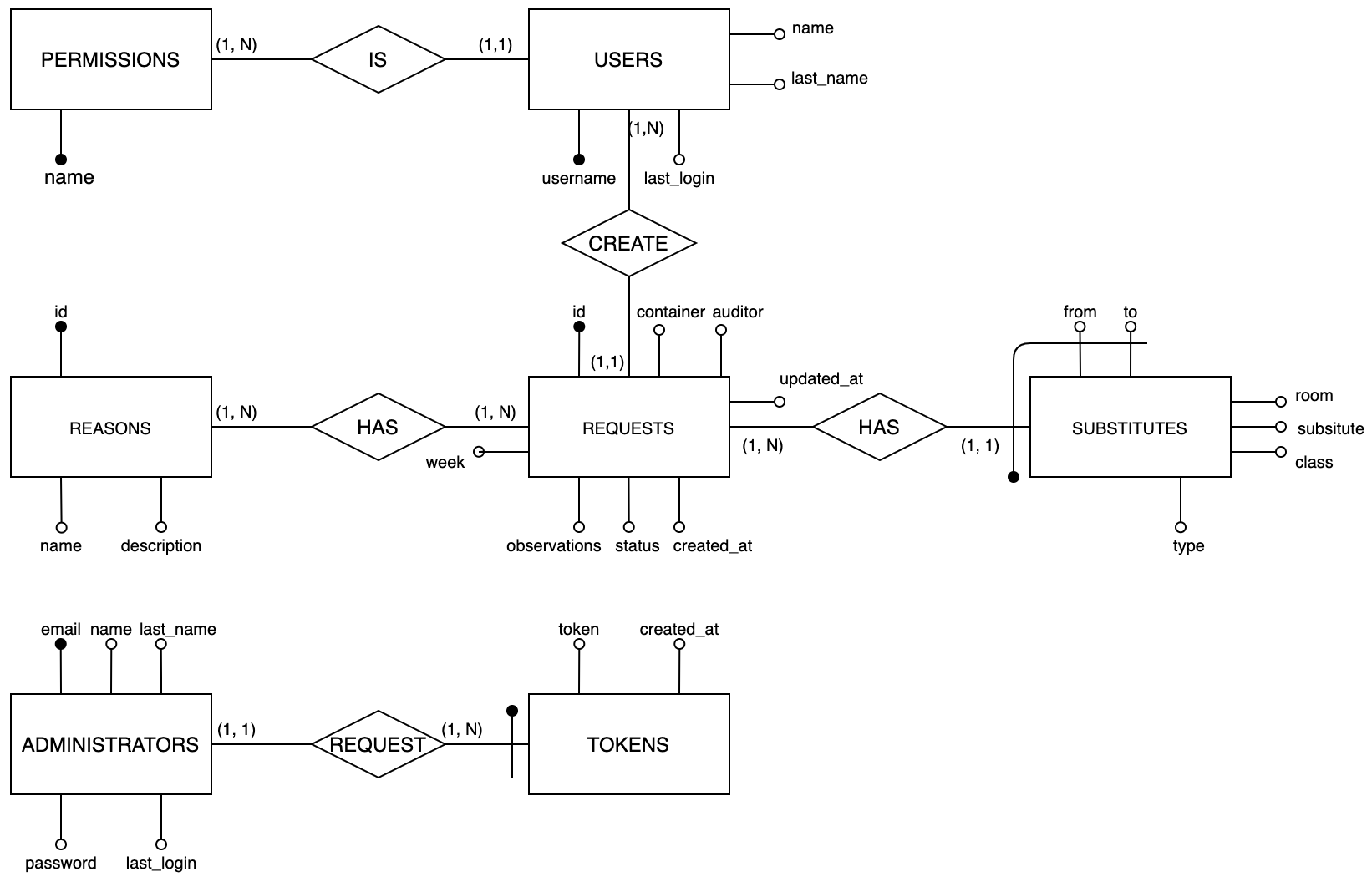


Figura 10 Schema ER banca dati.

Questo è lo schema ER utilizzato dall’applicativo web. È composto da sette tabelle principali ed una tabella ponte. La gestione degli amministratori del sito web avviene interrogando la tabella “ADMINISTRATORS” la quale contiene i dati relativi agli amministratori locali, è inoltre presente una tabella che si occupa della gestione dei token utilizzati per il recupero password degli utenti locali. La tabella “REASONS” contiene i motivi dei congedi che verranno correlati attraverso l’utilizzo di una tabella ponte con i congedi stessi salvati nella tabella “REQUESTS” la quale inoltre è collegata agli orari e possibili supplenti che vengono salvati nella tabella “SUBSTITUTES”. Gli utenti LDAP vengono salvati nella tabella “USERS” al primo login, questo permette di impostare il permesso di default, ovvero il docente.

DA\_REVISIONARE

### Descrizioni delle tabelle

|  |  |
| --- | --- |
| **PERMISSIONS** | |
| Attributo | Descrizione |
| name | Rappresenta il nome di un permesso all’interno del sistema. È un attributo di tipo stringa con un limite di 30 caratteri. Non può essere nullo e deve essere univoco.  Esempio: docente |

|  |  |
| --- | --- |
| **USERS** | |
| Attributo | Descrizione |
| username | Rappresenta l’identificativo di un utente che accede attraverso l’utilizzo di LDAP. È un attributo di tipo stringa con limite di 20 caratteri, non può essere nullo e deve essere univoco.  Esempio: filippo.finke |
| name | Rappresenta il nome di un amministratore. È un attributo di tipo stringa con un limite di 20 caratteri. Può contenere solamente lettere dell’alfabeto e non può essere nullo.  Esempio: Filippo |
| last\_name | Rappresenta il cognome di un amministratore. È un attributo di tipo stringa con un limite di 20 caratteri. Può contenere solamente lettere dell’alfabeto e non può essere nullo.  Esempio: Finke |
| last\_login | Rappresenta l’ultimo accesso dell’utente all’applicativo. È un attributo di tipo DATETIME e non può essere nullo.  Esempio: 2020-01-29 14:16:59 |

|  |  |
| --- | --- |
| **ADMINISTRATORS** | |
| Attributo | Descrizione |
| email | Rappresenta un indirizzo email dell’utente. È un attributo di tipo stringa con limite di 255 caratteri. Non può essere nullo e deve essere univoco.  Esempio: filippo.finke@samtrevano.ch |
| name | Rappresenta il nome di un amministratore. È un attributo di tipo stringa con un limite di 20 caratteri. Può contenere solamente lettere dell’alfabeto e non può essere nullo.  Esempio: Filippo |
| last\_name | Rappresenta il cognome di un amministratore. È un attributo di tipo stringa con un limite di 20 caratteri. Può contenere solamente lettere dell’alfabeto e non può essere nullo.  Esempio: Finke |
| password | Rappresenta la password di un amministratore. È un attributo di tipo stringa con limite di 255 caratteri. Non può essere nullo. All’interno di questo attributo verrà salvata un hash della password dell’utente.  Esempio: $2y$10$NmiaiLmr3dhUg3ePIExyt.l2KvE7SK6le1UH67QVikBlyBjjTHgVG |
| last\_login | Rappresenta l’ultimo accesso dell’utente all’applicativo. È un attributo di tipo DATETIME e non può essere nullo.  Esempio: 2020-01-29 14:16:59 |

|  |  |
| --- | --- |
| **TOKENS** | |
| Attributo | Descrizione |
| token | Rappresenta un codice che verrà utilizzato per eseguire il recupero della password. Questo codice verrà generato in modo casuale. All’interno dell’attributo verrà salvata un hash in SHA256 del token di recupero password. Il campo è di tipo stringa con un limite di 64 caratteri e non può essere nullo.  Esempio: 4b9eb466ad2615314c8194b1c46e6ef8e910d0b41a682e32de73503883e09b58 |
| created\_at | Rappresenta la data di creazione del codice di recupero password. È un attributo di tipo DATETIME e non può essere nullo.  Esempio: 2020-01-29 14:16:59 |

|  |  |
| --- | --- |
| **REASONS** | |
| Attributo | Descrizione |
| id | Rappresenta l’identificatore di una motivazione di congedo. È un attributo di tipo intero e viene impostato in modo automatico dall’applicativo. Non può essere nullo e deve essere univoco.  Esempio: 1 |
| name | Rappresenta il nome della motivazione. È un attributo di tipo stringa, ha un limite di 255 caratteri, può contenere solamente DA\_FARE e non può essere nullo.  Esempio: Assenza per attività fuori sede |
| description | Rappresenta la descrizione di una motivazione. È un attributo di tipo stringa, ha un limite di 255 caratteri, può contenere solamente DA\_FARE.  Esempio: Assenza per un attività fuori dalla sede scolastica. |

|  |  |
| --- | --- |
| **REQUESTS** | |
| Attributo | Descrizione |
| id | Rappresenta l’identificatore di una richiesta di congedo. È un attributo di tipo intero e viene impostato in modo automatico dall’applicativo. Non può essere nullo e deve essere univoco.  Esempio: 1 |
| status | Rappresenta lo stato della richiesta di congedo. È un attributo di tipo TINYINT senza segno.  Esempio: 0 |
| container | Rappresenta il contenitore nel quale si trova la richiesta. È un attributo di tipo TINYINT senza segno.  Esempio: 0 |
| week | Rappresenta la settimana nella quale è previsto il congedo. È un enum e può contenere “A” oppure “B”  Esempio: A |
| observations | Rappresenta le osservazioni correlate con la richiesta di congedo. È un attributo di tipo stringa con un limite di 255 valori, può contenere qualsiasi carattere e può essere nullo.  Esempio: Presumo che il docente X possa eseguire la supplenza. |
| auditor | Rappresenta il nome e il cognome dell’ultimo utente che ha eseguito una revisione al congedo.  Esempio: Filippo Finke |
| updated\_at | Rappresenta l’ultimo aggiornamento della richiesta di congedo. È un attributo di tipo DATETIME e non può essere nullo.  Esempio: 2020-01-29 14:16:59 |
| created\_at | Rappresenta la data di creazione della richiesto di congedo. È un attributo di tipo DATETIME e non può essere nullo.  Esempio: 2020-01-29 14:16:59 |

|  |  |
| --- | --- |
| **SUBSTITUTES** | |
| Attributo | Descrizione |
| from\_date | Rappresenta la data di inizio dell’assenza. È un attributo di tipo DATETIME e non può essere nullo.  Esempio: 2020-01-29 14:16:59 |
| to\_date | Rappresenta la data di fine dell’assenza. È un attributo di tipo DATETIME e non può essere nullo.  Esempio: 2020-01-29 14:16:59 |
| room | Rappresenta l’aula nella quale eseguire la supplenza. È un attributo di tipo stringa con limite di 5 caratteri e può essere nullo.  Esempio: A417 |
| substitute | Rappresenta il nome completo del docente di supplenza. È un attributo di tipo stringa con limite 30 caratteri e può essere nullo.  Esempio: Filippo Finke |
| class | Rappresenta la classe da sostituire. È un attributo di tipo stringa con limite 15 caratteri e può essere nullo.  Esempio: SAMT I4AC |
| type | Rappresenta il tipo di supplenza previsto. È un enum e può avere i seguenti valori:   * SI: supplenza interna * SO: scambio d’orario * SP: sorveglianza parallela * SE: supplente esterno   Esempio: SAMT I4AC |

DA\_REVISIONARE

### Schema logico

permissions(name)

users(username, name, last\_name, permission(FK), last\_login\*)

reasons(id, name, description)

requests(id, username(FK), week, container, status, observations\*, updated\_at, created\_at)

substitutes(request(FK), from\_date, to\_date, type\*, room\*, substitute\*, class\*)

request\_reason(request\_id(FK), reason\_id(FK))

administrators(email, name, last\_name, password, last\_login)

tokens(email(FK), token(UNIQUE), created\_at)

Questo è lo schema logico del database, è presente una tabella ponte chiamata “request\_reason” che permette di collegare i motivi di una richiesta di congedo ad un congedo.

DA\_REVISIONARE

## Design delle interfacce

### Pagina di accesso

Questo è un mockup della pagina utilizzata per accedere all’applicativo web. Attraverso questa pagina è quindi possibile inserire username e password per poter accedere tramite LDAP oppure il database locale. La funzionalità di recupero password verrà mostrata solamente quando l’utente sbaglierà la password e sarà un utente locale in quanto non è richiesto il reset di password anche per utenti LDAP. DA\_REVISIONARE



Figura 11 Pagina di accesso.

### Pagina principale

Questo è uno schizzo della pagina principale mostrata agli utenti che eseguono l’accesso attraverso LDAP. Attraverso questa pagina è quindi possibile riempire un modulo di congedo oppure navigare nei vari recipienti. Verranno mostrati solamente i recipienti relativi al permesso dell’utente corrente. Inoltre nella parte superiore destra verrà mostrato nome, cognome e il permesso dell’utente. DA\_REVISIONARE



Figura 12 Pagina principale.

### Pagina di un recipiente

Questo è lo schizzo della pagina di un recipiente, la quale sarà modificata dinamicamente in base ai recipienti. In questo caso viene mostrato un potenziale recipiente della segreteria. All’interno del recipiente sarà quindi possibile vedere tutte le richieste di congedo, con il loro stato e da chi sono state inviate. DA\_REVISIONARE



Figura 13 Pagina di un recipiente.

### Pagina storico

Questa è la pagina dello storico, all’interno di questa pagina vengono mostrati tutti i congedi registrati nel sistema. Inoltre nella parte superiore della pagina è possibile eseguire delle ricerche mirate. DA\_REVISIONARE



Figura 14 Pagina storico.

### Pagina amministrazione utenti

Attraverso questa pagina è possibile aggiungere utenti di tipo amministratore che potranno gestire l’applicativo web. È inoltre possibile modificare i permessi da assegnare agli utenti che eseguiranno l’accesso attraverso LDAP. DA\_REVISIONARE



Figura 15 Pagina amministrazione utenti.

### Pagina amministrazione congedi

Questa è la pagina di amministrazione della tipologia dei congedi. Attraverso questa pagina è dunque possibile modificare le motivazioni e/o cause dei congedi presenti nella pagina principale dell’applicativo. DA\_REVISIONARE



Figura 16 Pagina amministrazione congedi.

## Design dell’architettura del sistema

DA\_FARE

# Implementazione

## Gestione versioni

Come gestione di tutti i file che riguardano lo sviluppo del progetto, quindi documentazione, diari, codice e altro ho utilizzato una repository GitLab che è stata messa a disposizione sul server scolastico da parte dei formatori. Ho dunque utilizzato questa repository per tenere traccia di tutti i cambiamenti all’interno del progetto, i quali sono stati caricati su GitLab con descrizioni apposite in modo da permettere di tornare avanti oppure indietro nel corso del progetto.

DA\_REVISIONARE

## Gestore di pacchetti

Per lo sviluppo del progetto ho utilizzato delle librerie esterne. Per la gestione di pacchetti aggiuntivi di PHP ho utilizzato dunque Composer, il quale permette di gestire librerie esterne in modo semplificato. Ho quindi impostato Composer attraverso il file di configurazione “composer.json” nel seguente modo:

{

"name": "filippofinke/gestione-congedi",

"description": "Applicativo web per la gestione delle richieste di congedo dei docenti del CPT",

"type": "project",

"require": {

"filippofinke/php-rest": "dev-master"

},

"authors": [

{

"name": "Filippo Finke",

"email": "filippo.finke@samtrevano.ch"

}

],

"minimum-stability": "dev",

"autoload": {

"psr-4": {

"FilippoFinke\\":"src/"

}

}

}

Questo file mi permette dunque di importare la libreria “php-rest” la quale verrà utilizzata all’interno del progetto.

DA\_REVISIONARE

## Database

Il database è stato sviluppato in base allo schema ER. Le parti importanti nell’implementazione del database sono è stata la tabella ponte aggiuntiva per il collegamento della tabella “REASONS” e quella “REQUESTS”, la tabella è stata implementata nel seguente modo:

CREATE TABLE request\_reason(

request INT NOT NULL,

reason INT NOT NULL,

FOREIGN KEY(request) REFERENCES requests(id),

FOREIGN KEY(reason) REFERENCES reasons(id)

);

Questa tabella ha dunque i riferimenti alle tabelle “REASONS” e “REQUESTS”. Un altro fattore importante nell’implementazione della banca dati è stata la separazione della tabella “TOKENS” da “ADMINISTRATORS” questo per alleggerire il peso del database in quanto non tutti gli amministratori richiederanno un recupero password frequentemente. Ho dunque implementato la tabella separatamente nel seguente modo:

CREATE TABLE tokens (

email VARCHAR(255) NOT NULL,

token VARCHAR(64) UNIQUE NOT NULL,

created\_at DATETIME DEFAULT CURRENT\_TIMESTAMP NOT NULL,

FOREIGN KEY(email) REFERENCES administrators(email) ON UPDATE CASCADE ON DELETE CASCADE

);

Nella tabella “TOKENS” è quindi presente un riferimento alla tabella “ADMINISTRATORS” attraverso la colonna email.

Per eseguire la connessione al database utilizzando PHP mi sono basato sulla classe Database già presente nel framework php-rest, il codice per collegarsi è il seguente:

Database::setHost(DB\_HOST);

Database::setDatabase(DB\_NAME);

Database::setUsername(DB\_USERNAME);

Database::setPassword(DB\_PASSWORD);

$connection = Database::getConnection();

Vengono quindi impostati I parametri di connessione al database attraverso metodi appositi, successivamente si può ricavare la connessione da qualsiasi parte del codice con il metodo getConnection.

DA\_REVISIONARE

## Applicativo Web

### Struttura

L’applicativo è stato sviluppato con l’ausilio di php-rest e il template desk-app. La struttura del codice è la seguente:

.

├── src

│   ├── Assets

│   ├── Controllers

│   ├── Libs

│   ├── Middlewares

│   ├── Models

│   └── Views

└── vendor

La cartella src contiene tutto il codice dell’applicativo web. All’interno di questa cartella sono presenti ulteriori sottocartelle, la cartella Assets contiene tutte le risorse (come ad esempio i font, i fogli di stile, javascript, etc.), la cartella Controllers contiene tutte le classi che gestiscono gruppi di percorsi. La cartella Libs contiene classi utilizzate in generale all’interno del codice, come per esempio classi di gestione di sessione o di autenticazione. La cartella Middlewares contiene tutte le classi che si occupano di gestire gli accessi a determinati percorsi. All’interno di Models invece sono presenti tutte le classi utilizzate per interfacciarsi con i dati presenti nel sito web i quali verranno poi mostrati attraverso i controller nelle interfacce grafiche, le quali sono salvate nella cartella Views. È inoltre presente una cartella chiamata vendor, all’interno di questa cartella è salvato in modo automatico il codice generato da composer per la gestione e l’integrazione di librerie esterne (php-rest).

DA\_REVISIONARE

### Configurazione

L’applicativo web possiede un file di configurazione il quale permette di adattare dinamicamente alcune impostazioni. Attraverso questo file è possibile definire i parametri di connessione al database, l’indirizzo di posta elettronica dal quale verranno inviate le email, configurazione riguardanti l’interrogazione LDAP (server al quale collegarsi, prefisso del dominio, percorso di ricerca utenti e gruppi permessi all’accesso) e altre impostazione per dati dinamici come per esempio orari. All’interno del sorgente è presente un file di configurazione di esempio chiamato config\_sample.php, questo file dovrà essere rinominato in config.php una volta che il sistema verrà messo in produzione. Per caricare il file di configurazione all’interno dell’applicativo viene utilizzato il seguente codice:

// Controllo la presenza del file di configurazione.

if (!file\_exists("config.php")) {

exit("File di configurazione 'config.php' mancante! Puoi ricavarlo attraverso il file di esempio 'config\_sample.php!");

}

// Includo del file di configurazione dell'applicativo.

require \_\_DIR\_\_ . '/config.php';

Viene quindi controllata la presenza del file di configurazione da importare e se presente viene caricato, altrimenti la pagina stampa solamente un messaggio di informazione e poi termina l’esecuzione del codice.

DA\_REVISIONARE

### Autenticazione

Sono presenti due metodi di autenticazione all’interno dell’applicativo di gestione dei congedi, il primo metodo il quale sarà anche il più utilizzato è attraverso le credenziali del dominio scolastico, quindi autenticazione ed accesso attraverso LDAP. Il secondo metodo di autenticazione invece verrà utilizzato per la gestione del sito web e per eseguire l’accesso verrà interrogata una banca dati locale. Per determinare quale metodo di autenticazione per verificare le credenziali immesse da parte dell’utente verrà utilizzato il nome utente, il quale per l’autenticazione attraverso il database locale dovrà essere per forza un indirizzo di posta elettronica, mentre per l’autenticazione con LDAP tutt’altro. Il codice che si occupa di richiamare il metodo di autenticazione adeguato alle credenziali dell’utente è il seguente:

if (Validators::isValidEmail($username)) {

$user = LocalAuth::login($username, $password);

} else {

$user = Ldap::login($username, $password);

}

Il codice in questione è presente nel controller di autenticazione chiamato Auth. Questo codice si occupa dunque di controllare se il nome utente è un indirizzo email valido, in caso di riscontro positivo verrà tentato l’accesso attraverso l’autenticazione locale in caso contrario verrà provato l’accesso attraverso LDAP.

DA\_REVISIONARE

#### Autenticazione banca dati locale

Per eseguire l’accesso con credenziali locali è stata sviluppata una classe apposita. Questa classe si occupa di interrogare il database MySQL e di verificare l’esistenza dell’utente che sta provando ad accedere e della validità delle credenziali immesse. All’interno di questa classe chiamata LocalAuth è presente un solo metodo il quale si occupa di eseguire l’accesso, il metodo è chiamato login ed il codice contenuto è il seguente:

class LocalAuth

{

public static function login($email, $password, $bypass = false)

{

$user = Administrators::getByEmail($email);

if ($user && (password\_verify($password, $user["password"]) || $bypass))

{

if ($user["last\_login"] == null) {

$\_SESSION["force\_reset\_password"] = true;

}

return new Administrator($email, $user["name"], $user["last\_name"]);

}

return false;

}

}

Il metodo in questione si occupa in primo luogo di verificare la presenza dell’utente nella banca dati locale eseguendo una ricerca in base all’indirizzo di posta elettronica. Se l’utente in questione esiste viene eseguito un controllo con la password inserita nella pagina di login e quella presente nel database sotto forma di hash. Se i controlli vanno a buon fine viene controllato se l’utente deve eseguire un reset della password oppure no e successivamente viene ritornato un oggetto di tipo Administrator contenente i dati dell’utente. È inoltre presente un parametro bypass il quale viene utilizzato internamente nell’applicativo web per eseguire l’accesso senza l’utilizzo di una password.

DA\_REVISIONARE

#### Interfaccia LDAP

Per interfacciarsi con LDAP è stata sviluppata una classe apposita. Questa classe permette di verificare che le credenziali di un utente siano corrette attraverso un server LDAP. Inoltre questa classe permette di autorizzare solamente determinati utenti. La parte più importante di questa classe riguarda la gestione dell’accesso, è quindi presente un metodo login che accetta due parametri, username e password. Il metodo di controllo dell’accesso è stato implementato in questo modo:

public static function login($username, $password) {

$connectionString = ldap\_connect("ldap://".self::$host.":".self::$port."/");

ldap\_set\_option($connectionString, LDAP\_OPT\_PROTOCOL\_VERSION, 3);

ldap\_set\_option($connectionString, LDAP\_OPT\_REFERRALS, 0);

if ($connectionString) {

$bind = @ldap\_bind($connectionString, self::$prefix.$username, $password);

if ($bind) {

$filter = "(sAMAccountName=" . $username . ")";

$result = ldap\_search($connectionString, self::$dn, $filter, array());

$entries = ldap\_get\_entries($connectionString, $result);

if ($entries["count"] == 1) {

if (($entries[0]["useraccountcontrol"][0] & 2) == 2) {

// Account disabilitato.

return false;

}

$name = utf8\_encode($entries[0]["givenname"][0]);

$lastName = utf8\_encode($entries[0]["sn"][0]);

$groups = $entries[0]["memberof"];

foreach ($groups as $key => $value) {

if ($key != "count") {

if (self::isAllowed($value)) {

return new LdapUser($username, $name, $lastName);

}

}

}

// L'utente non ha il permesso.

return false;

} else {

// L'utente non è stato trovato.

return false;

}

} else {

// Errore username o password.

return false;

}

} else {

// Errore nella stringa di connessione al server LDAP.

return false;

}

}

Questo metodo si occupa dunque di validare in primo luogo che la stringa di connessione al server LDAP sia corretta, successivamente prova ad eseguire l’accesso al dominio con le credenziali che sono state passate come parametri. Se l’accesso tramite LDAP ha un esito positivo, viene ricavato l’utente corrente e ne vengono ricavati i permessi, i quali vengono successivamente controllati per stabilire se un utente può accedere oppure no all’applicativo.

DA\_REVISIONARE

#### Gestione delle sessioni e percorsi

Per gestire l’autenticazione dell’applicativo vengono utilizzate delle sessioni le quali mantengono lo stato dell’utente nelle varie pagine del programma. Per velocizzare e semplificare l’utilizzo delle sessioni è stata sviluppata una classe chiamata Session la quale permette di interfacciarsi con i metodi per la gestione delle sessioni del linguaggio in maniera semplificata. Per esempio per impostare la sessione di un utente come autenticato è presente il metodo authenticate il quale permette di salvare dati aggiuntivi nella sessione. Il codice del metodo è il seguente:

public static function authenticate($data = null)

{

$\_SESSION["authenticated"] = true;

if (is\_array($data)) {

foreach ($data as $key => $value) {

$\_SESSION[$key] = $value;

}

}

}

Questo codice si occupa dunque di impostare il valore della chiave authenticated a true all’interno della sessione, questo indice verrà poi utilizzato per controllare se un utente ha eseguito oppure no l’accesso all’applicativo con il metodo isAuthenticated:

public static function isAuthenticated()

{

return (isset($\_SESSION["authenticated"]) && $\_SESSION["authenticated"] == true);

}

Il metodo isAuthenticated si occupa dunque di controllare la presenza dell’indice e del suo valore. Inoltre in questa classe dedicata alla gestione delle sessioni è presente un metodo per eseguire la disconnessione:

public static function logout()

{

foreach ($\_SESSION as $key => $value) {

unset($\_SESSION[$key]);

}

session\_destroy();

}

Il metodo in questione si occupa di eliminare qualsiasi indice presente nella sessione e di distruggere definitivamente la sessione dell’utente. Sono presenti ulteriori metodi per ricavare lo stato del permesso di un utente, come per esempio il metodo isAdministrator il quale controlla se l’utente è un amministratore:

public static function isAdministrator()

{

return $\_SESSION["permission"] == "Administrator";

}

Questo metodo non fa altro che comparare l’indice del permesso dell’utente con la stringa Administrator per determinare se un utente è amministratore oppure no, sono presenti ulteriori metodi per ogni permesso all’interno della classe i quali vengono gestiti nella stessa maniera. Grazie a questi metodi è possibile dunque gestire i vari accessi alle varie pagine e percorsi presenti nell’applicativo web grazie all’utilizzo di classi o funzioni Middlewares, queste classi o funzioni permettono di eseguire del codice prima che un determinato percorso sia visitato, permettendo dunque di eseguire controlli sui permessi e decidere se un utente ha il permesso di accedere al percorso oppure no. Un esempio di Middleware è il seguente:

class AuthRequired

{

public function \_\_invoke($request, $response)

{

if (!Session::isAuthenticated()) {

$response->redirect('/login');

exit;

}

}

}

Questa classe si occupa di controllare se l’utente che sta visitando il percorso ha eseguito l’accesso all’applicativo attraverso la classe Session e il metodo isAuthenticated, se l’utente non ha eseguito l’accesso verrà rimandato alla pagina di login eseguendo un redirect, altrimenti verrà caricato il percorso normalmente. Sono presenti ulteriori classi simili le quali permettono di distinguere diverse condizioni di accesso, come per esempio verificare che un utente appartenga alla segreteria, che abbia eseguito il login con LDAP e così via. Per assegnare queste classi prima di un determinato percorso è possibile utilizzare i metodi predisposti dal framework php-rest before, per indicare di chiamare il metodo prima, oppure after, per indicare di chiamare il metodo dopo, quando viene istanziato un percorso. Per esempio:

// Pagina di visione e modifica congedi.

$router->get(

'/dashboard/{id:[0-9]+}',

'FilippoFinke\Controllers\Dashboard::index'

)

// Controllo che l'utente appartenga alla segreteria.

->before(new SecretaryRequired()),

In questo caso viene assegnato il controllo che l’utente appartenga alla segreteria per il percorso di visione e modifica dei congedi. I percorsi dell’applicativo vengono aggiunti attraverso l’oggetto Router il quale si occupa di eseguire la presenza di essi e quali funzioni o classi chiamare.

// Oggetto nel quale verranno salvati tutti i percorsi dell'applicativo.

$router = new Router();

Attraverso questo oggetto è quindi possibile aggiungere tutti i percorsi al quale l’applicativo dovrà rispondere:

$router->get('/percorso', 'Classe::metodo’);

$router->post('/percorso', 'Classe::metodo’);

$router->put('/percorso', 'Classe::metodo’);

$router->delete('/percorso', 'Classe::metodo’);

L’oggetto Router supporta percorsi di qualsiasi tipologia di richiesta, come per esempio: GET, POST, PUT, DELETE, etc. È possibile inoltre creare dei gruppi di percorsi:

// Gruppo di percorsi di gestione.

$adminRoutes = new RouteGroup();

// Aggiunta dei percorsi al gruppo.

$adminRoutes->add(

// Percorso per inserire una motivazione.

$router->post('/reasons', 'FilippoFinke\Controllers\Reasons::insert'),

. . .

// Percorso per l'eliminazione di un utente.

$router->delete('/users', 'FilippoFinke\Controllers\Users::delete')

)

// Aggiunta controllo autenticazione.

->before(new AuthRequired())

// Aggiunta controllo permesso amministratore.

->before(new AdministratorRequired());

Come ad esempio in questo caso nell’applicativo viene creato un gruppo dedicato ai percorsi di gestione di esso in modo da applicare i controlli di permessi al gruppo stesso.

DA\_REVISIONARE

### Interfaccia invio email

Per rendere più semplice e veloce l’invio di email all’interno dell’applicativo è stata sviluppata una classe dedicata. La classe in questione si chiama “Mail”, è presente un solo metodo chiamato send. Il metodo in questione si occupa di inviare un messaggio di posta elettronica ad un indirizzo email con un soggetto e un contenuto Opzionalmente la classe ha anche la possibilità di inviare degli allegati. Il metodo è stato implementato nel seguente modo:

public static function send($to, $subject, $message, $file = null, $filename = null)

{

$separator = md5(time());

$eol = "\r\n";

$headers = "From: name <".self::$fromEmail.">" . $eol;

$headers .= "MIME-Version: 1.0" . $eol;

$headers .= "Content-Type: multipart/mixed; boundary=\"" . $separator . "\"" . $eol;

$headers .= "Content-Transfer-Encoding: 7bit" . $eol;

$headers .= "This is a MIME encoded message." . $eol;

$body = "--" . $separator . $eol;

$body .= "Content-Type: text/html; charset=\"utf-8\"" . $eol . $eol;

$body .= $message . $eol . $eol;

if ($file && $filename) {

$content = chunk\_split(base64\_encode($file));

$body .= "--" . $separator . $eol;

$body .= "Content-Type: application/octet-stream; name=\"" . $filename . "\"" . $eol;

$body .= "Content-Transfer-Encoding: base64" . $eol;

$body .= "Content-Disposition: attachment" . $eol;

$body .= $content . $eol . $eol;

$body .= "--" . $separator . "--";

}

return mail($to, $subject, $body, $headers);

}

Il metodo si occupa di generare dinamicamente gli header dello standard relativo alla posta elettronica in base ai parametri che sono stati selezionati nella chiamata del metodo. In questo modo viene costruito il contenuto dell’email il quale può essere inviato. Grazie a questo classe basterà chiamare questo metodo statico da qualsiasi altra parte del codice per inviare una email.

DA\_REVISIONARE

### Validazione campi

La validazione dei campi è indispensabile per non avere delle incongruenze con i dati che verranno salvati all’interno della banca dati. I controlli di validazione e risanamento dei dati vengono eseguiti sia lato client, quindi da parte del browser, sia lato server. Nella parte lato server sono presenti dei metodi di verifica scritti in PHP all’interno della classe Validators la quale racchiude tutti i metodi di questo tipo. Ad esempio per verificare che la descrizione o le osservazioni di un congedo siano valide è presente il metodo isValidDescription il quale è stato implementato nel seguente modo:

public static function isValidDescription($text, $min = 1, $max = 255)

{

$safe = htmlspecialchars($text);

return strlen($text) >= $min && strlen($text) <= $max && $safe == $text;

}

Il metodo si occupa dunque di prevenire attacchi di tipo XSS e controllare che il testo abbia una lunghezza tra il minimo e il massimo specificati come parametro. Lo stesso metodo è presente sotto forma di funzione per il client, il codice è stato portato su JavaScript:

function isValidDescription(text, min = 1, max = 255) {

return text.length >= min && text.length <= max;

}

Ovviamente sono presenti ulteriori metodi di validazione, come per esempio la validazione del nome, del cognome, dell’email e molti altri. Per ogni metodo di verifica lato server scritto in PHP è presente un metodo trascritto in JavaScript per la parte lato client, vengono eseguiti controlli anche lato client per diminuire il carico del server.

DA\_REVISIONARE

### Risorse locali

L’applicativo ha bisogno dell’utilizzo di risorse locali scritte in CSS, JavaScript, font ed immagini per funzionare correttamente, le risorse utilizzate si trovano all’interno della cartella Assets nel codice sorgente. Per servire queste risorse agli utenti che visitano il sito web è presente un controller che si occupa della gestione di esse. Sono registrati quattro percorsi relativi alle risorse nel router delle richieste, i percorsi sono i seguenti:

// Percorso per file javascript.

$router->get('/assets/js/{asset}', 'FilippoFinke\Controllers\Assets::js');

// Percorso per file css.

$router->get('/assets/css/{asset}', 'FilippoFinke\Controllers\Assets::css');

// Percorso per i font.

$router->get('/assets/fonts/{asset}', 'FilippoFinke\Controllers\Assets::fonts');

// Percorso per le immagini.

$router->get('/assets/img/{asset}', 'FilippoFinke\Controllers\Assets::img');

Il controller che si occupa delle risorse è anch’esso chiamato Assets e si trova nella cartella Controllers, questa classe si occupa di recuperare le risorse locali, verificarne l’esistenza e servirle all’utente. All’interno del Controller sono definite le costanti che rappresentano i percorsi dal quale ricavare i dati da servire, ad esempio la cartella base e le risorse relative a JavaScript sono scritte nel seguente modo:

// Cartella base delle risorse.

private const ASSETS\_FOLDER = \_\_DIR\_\_ . '/../Assets';

// Cartella per i file JavaScript.

private const JS\_FOLDER = self::ASSETS\_FOLDER . '/js/';

Questo permette di rendere dinamica la ricercar delle risorse qualora si cambi cartella principale. Per ogni tipologia di risorsa è presente un metodo che si occupa di ricavare quale file è stato richiesto e la sua tipologia, ad esempio per le risorse di tipo CSS il metodo chiamato è il seguente:

public static function css($request, $response)

{

$asset = $request->getAttribute('asset');

return self::handle(self::CSS\_FOLDER.$asset, $response, 'text/css');

}

Questo metodo si occupa principalmente di definire il tipo di dato che sarà dato in risposta alla richiesta dell’utente. Il tutto viene delegato ad un metodo ulteriore chiamato handle il quale si occupa di gestire tutti i controlli e di servire il dato. Il metodo in questione è implementato nel seguente modo:

private static function handle($file, $response, $type = null)

{

$must = realpath(self::ASSETS\_FOLDER);

$realPath = realpath($file);

if (strpos($realPath, $must) !== false && file\_exists($file)) {

if (!$type) {

$type = mime\_content\_type($file);

}

return $response

->withHeader("Content-Type", $type)

->withBody(file\_get\_contents($file))

->withStatus(200);

} else {

return $response->withStatus(404)->withText("Not found");

}

}

Il metodo handle si occupa di ricavare il percorso reale della cartella base di tutte le risorse e lo confronta con il file che è stato richiesto, questo per prevenire attacchi di tipo Directory Traversal. Inoltre viene controllata l’esistenza del file richiesto, se presente viene controllato il tipo di file da servire nel caso non sia stato specificato nella chiamata del metodo e successivamente ne viene letto il contenuto il quale viene inviato all’utente in risposta alla richiesta. In caso la risorsa richiesta non sia disponibile, viene ritornata una risposta segnalando che il file non è stato trovato (404).

DA\_REVISIONARE

### Calendario

IN\_CORSO

DA\_REVISIONARE

### Sicurezza

La sicurezza è molto importante in questo progetto, questo dovuto al fatto che l’applicativo conserverà dati sensibili e sarà inoltre accessibile da reti esterne a quelle scolastiche.

DA\_REVISIONARE

#### Password locali

La password degli utenti locali (amministratori) viene salvata all’interno del database sotto forma di un hash generata utilizzando l’algoritmo bcrypt. Viene utilizzata la funzione password\_hash di PHP con il parametro PASSWORD\_DEFAULT, questo in modo tale che se bcrypt diventasse obsoleto e PHP scegliesse un altro algoritmo da utilizzare non siano necessari cambiamenti di codice. Il codice utilizzato per generare un hash della password è il seguente:

$hash = password\_hash($password, PASSWORD\_DEFAULT);

#### Recupero password

Il recupero della password viene eseguito attraverso l’uso di una stringa, definita token, con una lunghezza di 20 caratteri. Questo token viene generato in modo randomico dall’applicativo e viene salvato in correlazione con l’utente che ha richiesto il recupero della password assieme alla data di creazione della stringa. Questo token viene salvato all’interno del database utilizzando una sha256, questo in modo tale che anche se un utente abbia accesso al database in lettura non possa resettare le password degli utenti. Inoltre quando viene salvato questo token all’interno della banca dati viene salvata anche la sua data di creazione. I token hanno una vita massima di 10 minuti, una volta scaduti il token verrà disattivato. La generazione del token viene eseguita nel seguente modo:

private static function generateToken()

{

$keyspace = '0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ';

$pieces = [];

$max = mb\_strlen($keyspace, '8bit') - 1;

for ($i = 0; $i < 20; ++$i) {

$pieces []= $keyspace[random\_int(0, $max)];

}

return implode('', $pieces);

}

Viene assegnato un set di caratteri da utilizzare per la creazione del token. Da questo set vengono ricavati dei caratteri da posizioni casuali in modo da formare una stringa di 20 caratteri massimo.

Per controllare la validità e la presenza del token viene utilizzata questa funzione:

public static function login($token)

{

$hash = hash('sha256', $token);

$pdo = Database::getConnection();

$query = "SELECT \* FROM tokens WHERE token = :token";

$stm = $pdo->prepare($query);

$stm->bindParam(':token', $hash);

try {

$stm->execute();

$token = $stm->fetch(\PDO::FETCH\_ASSOC);

if ($token && time() - strtotime($token["created\_at"]) <= 60 \* 10) {

$\_SESSION["force\_reset\_password"] = true;

$user = LocalAuth::login($token["email"], false, true);

$user->updateLastLogin();

Session::authenticate(array(

"username" => $user->getUsername(),

"name" => $user->getName(),

"lastName" => $user->getLastName(),

"permission" => $user->getPermission()

));

self::resetTokens($token["email"]);

return true;

}

return false;

} catch (PDOException $e) {

return false;

}

}

Questa funzione login accetta un token come parametro, come prima cosa viene eseguita una sha256 del token inserito dall’utente in modo da poterlo comparare con i dati presenti nel database. Una volta eseguita l’hash viene interrogato il database per la ricerca di record con questo token, se è presente all’interno del database viene controllata la data di creazione e che sia minore di 10 minuti dal momento nel quale è stato utilizzato, se entrambi i requisiti sono soddisfatti l’utente può accedere all’applicativo. Una volta eseguito l’accesso verrà chiesto all’utente di cambiare la password. Se invece il token è inesistente o scaduto l’utente verrà rimandato alla pagina di accesso.

DA\_REVISIONARE

# Test

## Protocollo di test

DA\_FARE

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

## Risultati test

DA\_FARE

## Mancanze/limitazioni conosciute

DA\_FARE

# Consuntivo

DA\_FARE

# Conclusioni

DA\_FARE

## Sviluppi futuri

DA\_FARE

## Considerazioni personali

DA\_FARE

# Bibliografia

## Sitografia

DA\_FARE

# Glossario

|  |  |
| --- | --- |
| **Parola** | **Descrizione** |
| LDAP | Lightweight Directory Access Protocol, è un protocollo per l’interrogazione e la modifica di servizi di directory. |
| PDF | Portable Format Document, è un formato di file utilizzato per la rappresentazione di testo ed immagini sviluppato da Adobe. |
| REST | Representational State Transfer, è uno stile di sviluppo di software. |
| frontend | Interfaccia accessibile da parte degli utenti, la parte grafica. |
| backend | Parte di un software che elabora i dati generati dal frontend. |
| mockup | Rappresentazione di interfacce grafiche in modo generale. |
| MySQL | È un software per la gestione di database. |
| PHP | Hypertext Preprocessor, linguaggio di programmazione lato server utilizzato spesso nello sviluppo web. |
| Composer | Gestore di pacchetti aggiuntivi per il linguaggio PHP. |
| hash | È il risultato di una funzione di hash, un algoritmo che permette di generare una stringa di lunghezza fissa composta da caratteri casuali. |
| bcrypt | È una funzione di hash. |
| middlewares | Funzioni o classi che fungono da intermediari. |
| FPDF | FPDF è una classe PHP che permette di generare file PDF. |
| CSS | DA\_FARE |
| JavaScript | DA\_FARE |

DA\_REVISIONARE

# Allegati

DA\_FARE

DA\_FARE = Cose ancora da fare.

DA\_REVISIONARE = Cose quasi finite, da revisionare.

IN\_CORSO = Capitolo in scrittura.