Knowledge Base

Titolo del progetto: Knowledge Base
Alunno/a: Bryan Beffa
Classe: Info I4AA
Anno scolastico: 2019/2020
Docente responsabile: Ivan Raimondi

1 Introduzione

1.1 Informazioni sul progetto

Allievo coinvolto nel progetto: Beffa Bryan
Classe: Informatica 4AA presso la sede Scuola Arti e Mestieri Trevano
Docenti responsabili: Ivan Raimondi
Data inizio: 03:09:2019
Data fine: 20:12:2019

1.2 Abstract

The teachers need a website application with a login system. In this web application, the users can view cases of technical documentation and sort them using filters. The cases can be filtered by category, last entered, most recurrent cases. There must be two types of users: the first is the administrator who can add or remove users using a “user management” page. The admin can also add, remove or modified cases and categories as well. The second type of users are the operators who can just view and filter cases and create a new case.

1.3 Scopo

Mi è stato richiesto di realizzare un applicativo web per la gestione di casi di documentazione tecnica per un’azienda di supporto informatico. Nell’applicativo devono essere presenti due tipi di utenti: amministratore e operatori (utenti base). L’amministratore possiede più privilegi che comprendono l’aggiunta, la modifica e l’eliminazione di utenti, la modifica, eliminazione e registrazione dei casi di documentazione tecnica, aggiunta ed eliminazione di una categoria. L’amministratore è l’unico tipo di utente che può raggiungere la pagina di gestione degli utenti. Gli operatori non possono registrarsi, hanno i privilegi per registrare nuovi casi di documentazione tecnica e possono visualizzare tutti i casi effettuando delle ricerche applicando alcuni filtri di ricerca. Solo gli amministratori possono effettuare le modifiche dei dati personali di un utente base.

1.4 Analisi

1.5 Analisi del dominio

Mi è stato richiesto di creare un applicativo web per gestire casi di documentazione di una ditta di informatica. Nel sistema sono definiti 2 tipi di utenti, ovvero amministratore ed operatori. Deve esserci la possibilità di effettuare delle ricerche relativi ai casi registrati. In base al tipo di utente si hanno diversi tipi di permessi. Gli operatori (utenti con privilegi base) possono inserire dei nuovi casi, ma non hanno la possibilità di eliminarli o modificarli. L'amministratore può registrare nuovi casi, eliminarli e modificarli. Inoltre possono aggiungere, eliminare gli utenti e modificare le loro informazioni personali. Possono anche eliminare ed aggiungere categorie. L'applicativo web deve essere user friendly, di facile utilizzo e con una grafica apprezzabile.

1.6 Analisi e specifica dei requisiti

Il committente richiede un applicativo web per la gestione di casi di documentazione tecnica per la propria azienda. Mi è stato richiesto di creare due tipi di utenza, l'amministratore, l'unico utente che può eliminare e modificare i casi di documentazione tecnica, e gli operatori (utenti con privilegi base), che possono visualizzare e registrare nuovi casi nel sistema. L'amministratore può svolgere tutte le funzioni, comprese quelle degli operatori. Deve essere presente un sistema di ricerca dei vari casi, che permette la ricerca e successivamente la visualizzazione dei risultati. Si devono poter visualizzare gli ultimi casi, i casi più ricorrenti ed in base alla ricerca effettuata dall'utente.
Vi deve essere anche un sistema di registrazione per i nuovi utenti alla quale solo l'admin può avere accesso, un sistema di login ed un sistema di cambio delle loro informazioni personali (compresa la password). L'applicativo deve essere un sistema user friendly, apprezzabile a livello grafico e di semplice utilizzo. Ogni caso appartiene ad una categoria, possiede la data di prima apparizione ed il numero di rappresentazioni. I casi possono avere delle varianti. Gli utenti admin possono inoltre aggiungere o eliminare le categorie.

ID: REQ-01	
Nome	Pagina di login
Priorità	1
Versione	1.0
Note	La pagina deve presentare una mascherina per il login
Sotto requisiti	
001	Campo testuale per inserire l'indirizzo email
002	Campo testuale nascosto per inserire la password
003	Bottone "accedi" per effettuare il login
004	Messaggio di errore credenziali sbagliate
005	Messaggio di errore per problemi di connessione al database

ID: REQ-02	
Nome	Pagina di gestione utenti (admin)
Priorità	1
Versione	1.0
Note	Si necessita una pagina di gestione degli utenti alla quale solo l'admin può accedere
Sotto requisiti	
001	Possibilità di inserire nuovo utente
002	Possibilità di eliminare un utente
003	Possibilità di richiedere un cambio password per un utente
004	Per ogni azione richiesta di conferma

ID: REQ-03	
Nome	Pagina visualizzazione casi
Priorità	1
Versione	1.0
Note	Pagina in cui si possono visualizzare i casi
Sotto requisiti	
001	Maschera in cui si possono effettuare le varie ricerche relative ai casi esistenti
002	Bottone "aggiungi un caso" per registrare un nuovo caso
003	Bottone "elimina caso" (solo amministratore)
004	Bottone "modifica caso" (solo amministratore)
005	Bottone "elimina categoria" (solo amministratore)
006	Bottone "aggiunta categoria" (solo amministratore)

ID: REQ-04	
Nome	Pagina di registrazione casi
Priorità	1
Versione	1.0
Note	Pagina che permetta la registrazione di un nuovo caso
Sotto requisiti	
001	Campo testuale titolo
002	Campo testuale per la descrizione del caso
003	Possibilità di aggiungere una nuova categoria
004	Input per assegnare una categoria al caso

ID: REQ-05	
Nome	Ricerca casi
Priorità	1
Versione	1.0
Note	Sistema di ricerca casi di documentazione tecnica
Sotto requisiti	
001	Possibilità di visualizzare ultimi casi
002	Possibilità di visualizzare casi più ricorrenti
003	Bottone “registra nuovo caso”
004	Bottone registra nuova categoria (admin)
005	Bottone “logout” per effettuare la disconnessione del account corrente

ID: REQ-06	
Nome	Sistema di registrazione
Priorità	1
Versione	1.0
Note	Scrittura del codice lato server per il sistema di registrazione di un utente
Sotto requisiti	
001	Pagina di registrazione e database

ID: REQ-07	
Nome	Sistema di login
Priorità	1
Versione	1.0
Note	Scrittura del codice lato server per il sistema di login
Sotto requisiti	
001	Pagina di login e database

ID: REQ-08	
Nome	Sistema di aggiunta/eliminazione categoria (admin)
Priorità	1
Versione	1.0
Note	L’admin può aggiungere o eliminare una categoria
Sotto requisiti	
001	Campo testuale in cui inserire il nome della categoria
002	Bottone per eliminare una categoria

1.7 Caso d'uso

Questo schema rappresenta il caso d'uso dell'applicativo web evidenziando i vari ruoli ed il loro comportamento. Come si può notare ogni viene mostrato il soggetto (admin, operatori, database, sistema) e le rispettive funzionalità oltre al come sono collegate tra loro.

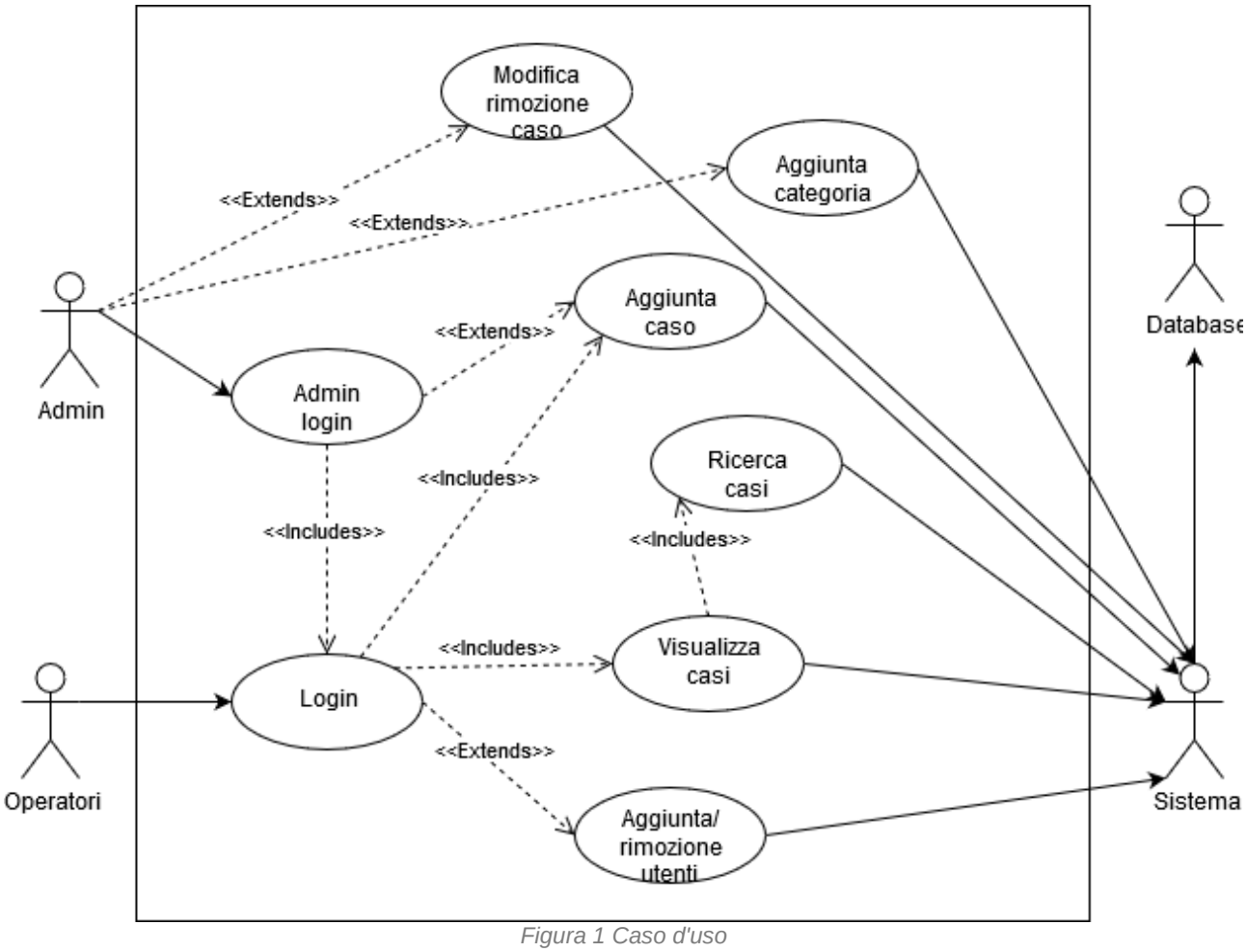


Figura 1 Caso d'uso

1.8 Pianificazione

Ho diviso il progetto in 4 fasi principali. La prima fase è quella di progettazione dove vengono svolte attività incentrate sull'analisi del progetto. La seconda fase è quella di sviluppo, dove avviene la stesura effettiva del codice delle pagine, script. La terza attività è il momento in cui vengono testati le varie parti di progetto. L'ultima fase, che copre tutta la durata del progetto, è la scrittura della documentazione. Ho utilizzato un diagramma di Gantt per gestire il mio progetto.

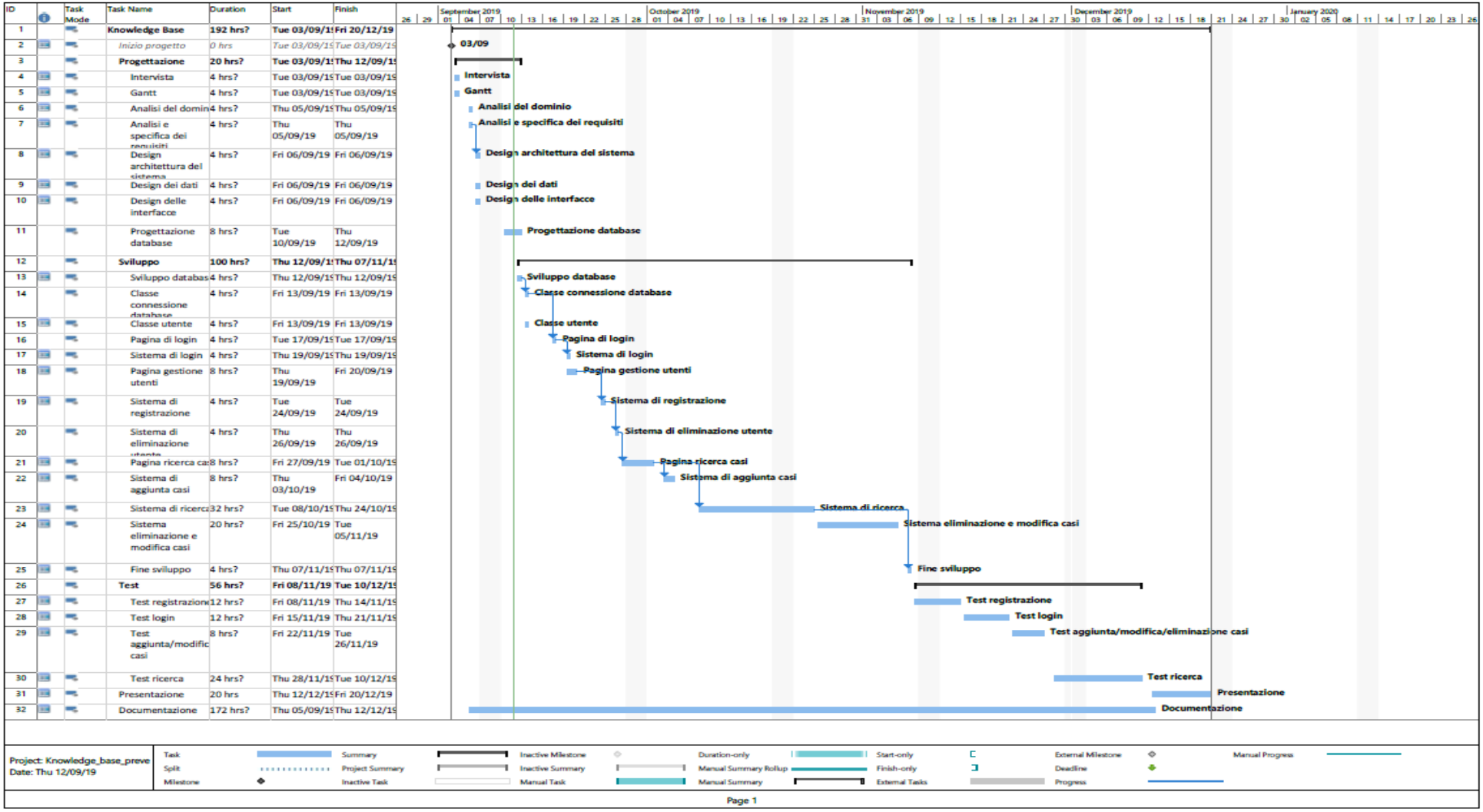


Figura 2 Diagramma di Gantt

1.9 Analisi dei mezzi

1.9.1 Software

I software che sono stati utilizzati per la realizzazione di questo progetto sono:

- JetBrains PhpStorm 2018.3.4
- Microsoft Word 2015
- Microsoft Powerpoint 2015
- Microsoft Project 2016 Professional
- XAMPP 7.3.9
- Php 7.1.33
- XDebug 2.6.1
- MySQL 15.1
- MySQL Workbench 8.0.17 CE

Il progetto è stato testato sui seguenti browser:

- Google Chrome 78.0.0
- Microsoft Edge 14.11.2019 → alcuni problemi di grafica ma nessuna conseguenza sulla funzionalità
- Mozilla Firefox 70.0.1 → animazione di caricamento ferma, il resto funziona correttamente

1.9.2 Hardware

L'intero progetto è un sito web, quindi non necessitavo di hardware specifico ed ho eseguito il tutto sul mio computer portatile Asus con sistema operativo windows 10.

2 Progettazione

In questo capitolo è rappresentato e spiegato come ho deciso di progettare l'intero sistema. Ho progettato tutto il sistema nel modo che ritenevo più efficace e coerente per garantire un prodotto che superasse le aspettative.
Ho cercato di rendere la navigazione e la struttura delle pagine il più semplice possibile e pulita possibile.

2.1 Design dell'architettura del sistema

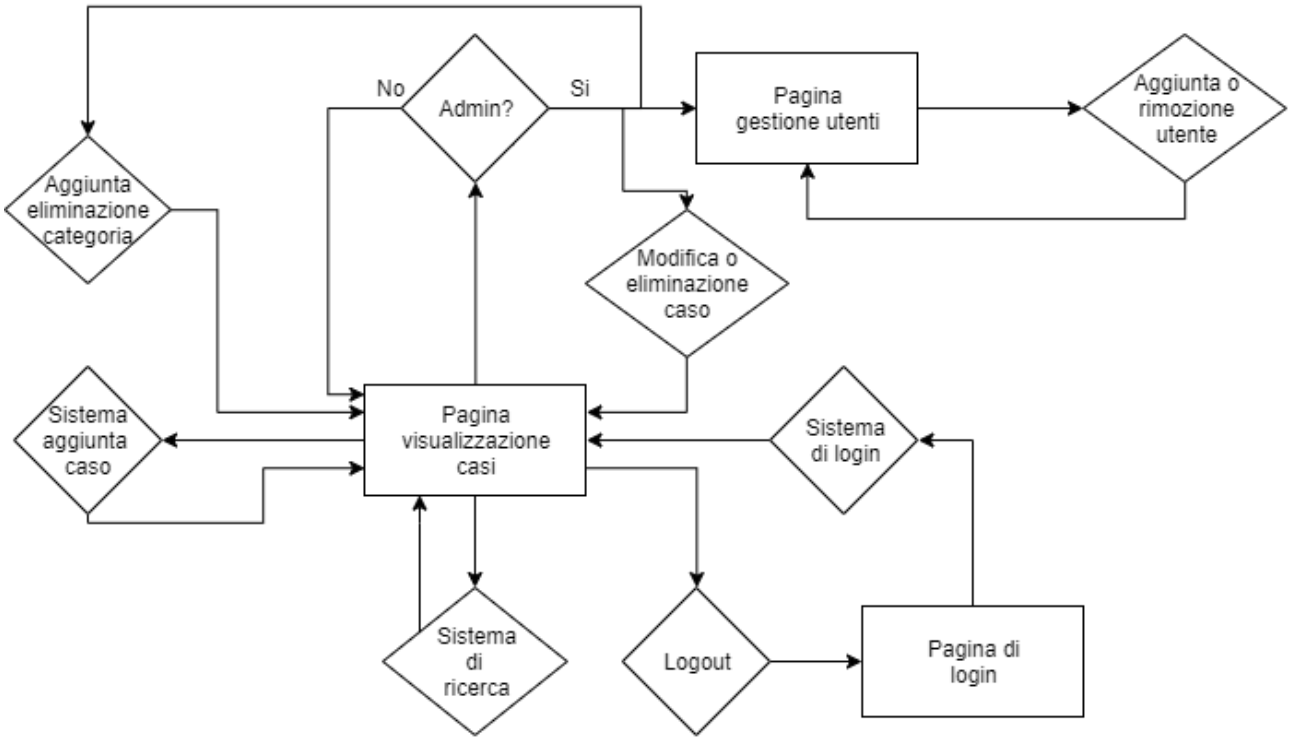


Figura 3 Design dell'architettura

L'utente parte dalla pagina di login, dove può effettuare l'accesso se è in possesso di credenziali corrette. Una volta effettuato l'accesso con delle credenziali valide l'utente viene reindirizzato alla pagina di visualizzazione casi.

Qui l'utente può effettuare delle ricerche tramite il sistema di ricerca casi. In questa pagina possono essere aggiunti nuovi casi.

Se l'utente possiede i privilegi amministrativi, può aggiungere ed eliminare le categorie. Inoltre può accedere alla pagina di gestione utente, in cui ha la possibilità di aggiungere, eliminare utenti.

2.2 Design dei dati e database

Dopo aver analizzato gli elementi che devono essere gestiti all'interno dell'applicativo ho progettato questo schema in cui sono presenti quattro tabelle **users, case, category** e **rappresentations**.
Questo database può essere modificato ed essere dettagliato da chiunque prenda in mano questo progetto in un secondo momento, così da poter rendere l'applicativo ancora più completo.

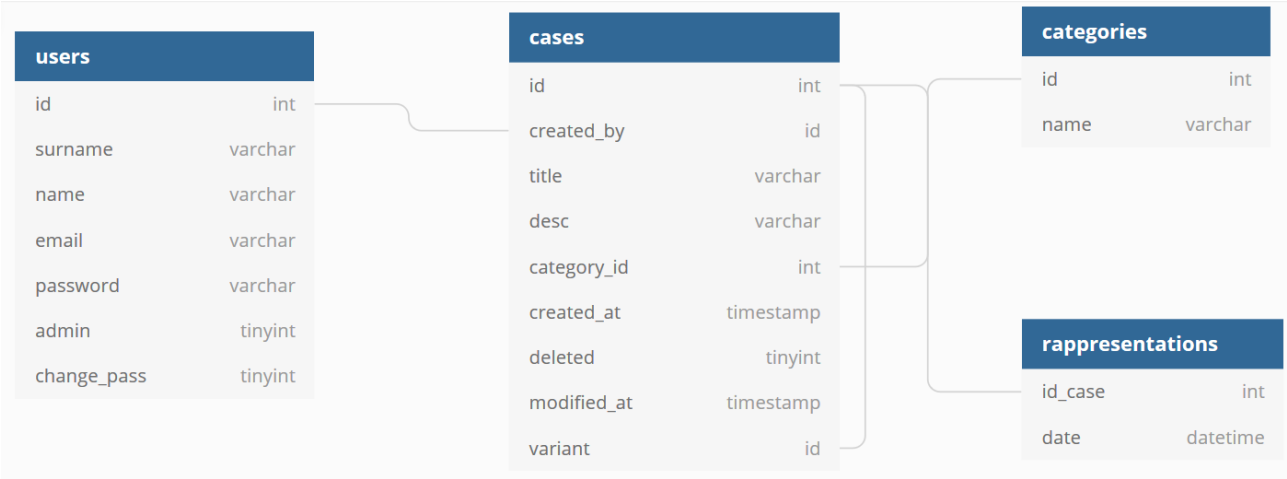


Figura 4 Schema del database

2.2.1 Tabella users

Nella tabella **users** ho inserito gli attributi che definiscono un modello di utente. Per gli utenti ho inserito solo le informazioni principali che mi servono all'interno del applicativo.

Attributo	Descrizione
Id	Numero identificativo dell'utente, il tipo di dato è un int
Name	Nome dell'utente, il tipo di dato è varchar
Surname	Cognome dell'utente, il tipo di dato è varchar
Email	Email dell'utente, varchar
Password	Password criptata dell'utente, varchar
Change_pass	Tinyint, definisce è stato richiesto un cambio di password dell'utente
Admin	Tinyint, definisce se un utente è admin o meno

2.2.2 Tabella cases

La tabella **cases** definisce il modello di “caso di documentazione tecnica”. Ogni caso deve avere i seguenti attributi.

Nella tabella seguente vi sono gli attributi e la loro descrizione:

Attributo	Descrizione
Id	Numero identificativo del caso, il tipo di dato è un int
Created_by	Foreign key, definisce l’utente che ha registrato il caso
Title	Tipo di dato varchar, indica il titolo del caso
Descr	Descrizione del caso
Category_id	Foreign key che definisce la categoria del caso
Created_at	Timestamp, indica la data e l’ora della creazione del caso
Deleted	Indica se il caso deve essere visualizzato dagli utenti
Modified_at	Timestamp, quando è stato modificato il caso
Variant	Int, id del caso a cui fa riferimento, campo non obbligatorio

2.2.3 Tabella categories

La tabella **categories** definisce un modello di “categoria”. Per ogni categoria vi è solo il numero identificativo della categoria ed il suo nome.

Attributo	Descrizione
Id	Numero identificativo category, il tipo di dato è un int
Name	Nome della categoria, tipo di dato varchar

2.2.4 Tabella representation

La tabella **representation** è un modello di rappresentazione, ogni caso viene salvato all’interno di questa tabella in modo da sapere ogni caso quante volte è apparso e quando.

Se un caso ha una variante, questa viene inserita all’interno della tabella con la data di apparizione.
Questa tabella è pensata per un’implementazione futura in cui gli utenti di tipo admin possano vedere quando sono stati aggiunti e creati casi che possiedono una variante.

Attributo	Descrizione
Id_case	Id del caso
Date	Timestamp, data e ora di quando è stato registrato il caso
Id_variant	Id della variante

Nota: Durante lo sviluppo del progetto mi sono accorto che questa tabella non era necessaria per le funzionalità richieste, ma in futuro può essere utilizzata per salvare la date delle modifiche. Per il momento è utilizzata solo per salvare i casi che vengono registrati con la relativa variante.

2.3 Design delle interfacce

2.3.1 Pagina di login

La pagina di login è molto semplice, presenta un campo testuale per inserire il nome dell'utente, un campo testuale nascosto per inserire la password ed infine il bottone "accedi".

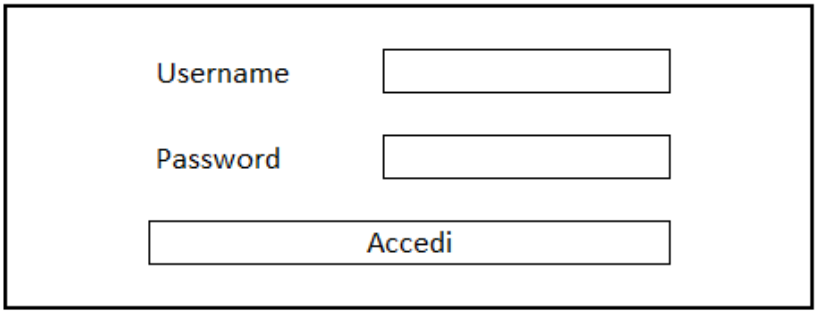


Figura 5 Design interfaccia di login

2.3.2 Pagina di gestione utenti

La pagina di gestione utenti è raggiungibile solo dagli utenti amministratori. La pagina presenta una struttura molto semplice, una grigli in cui si possono visualizzare tutte le informazioni dei utenti.
L'unico valore non visualizzabile è la richiesta di password, al suo posto vi è un bottone per (campo "richiesta cambio password") che serve ad effettuare una richiesta di cambio password.

Figura 6 Design interfaccia gestione utenti

2.3.3 Pagina di visualizzazione casi

La pagina di visualizzazione casi permette a tutti gli utenti (admin ed operatori) di visualizzare i vari casi di documentazione tecnica. A questa pagina possono accedere sia gli operatori che l'admin. L'admin ha la possibilità di modificare o cancellare un caso.

Layout admin:

Possibilità di eliminare e modificare un caso.

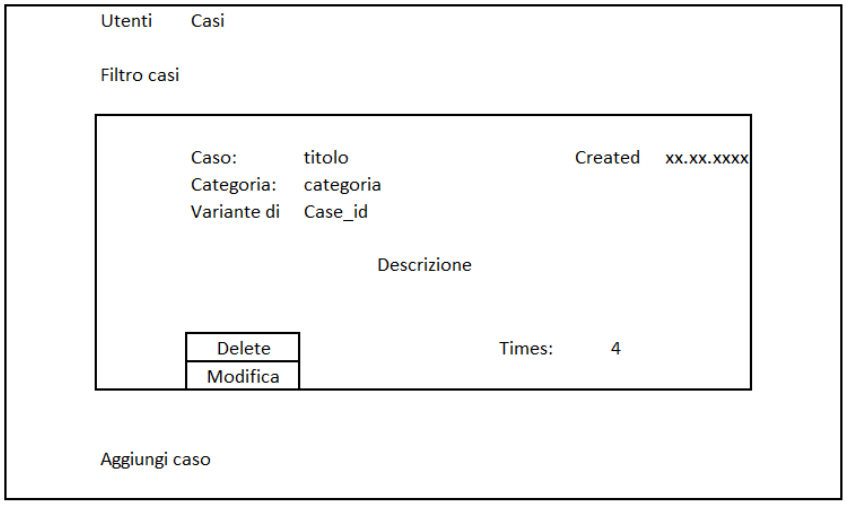


Figura 7 Layout admin

Layout operatori:

Il layout è uguale a quello dell'admin, ma alcune funzioni non sono presenti.

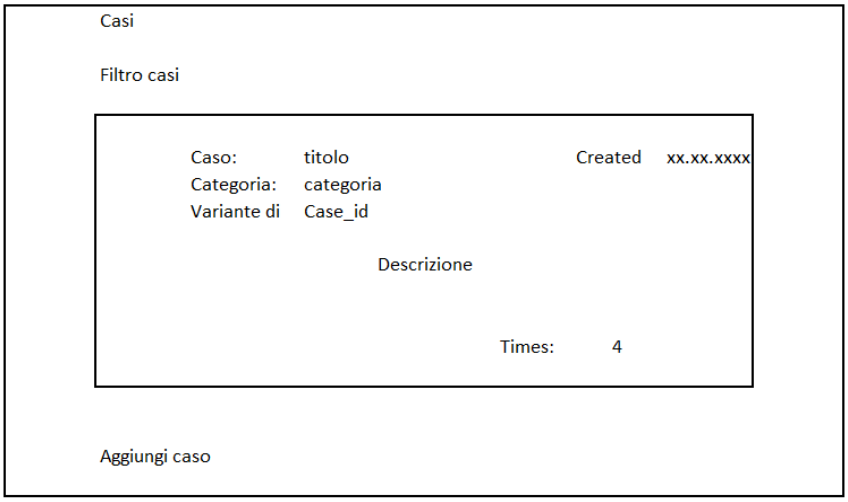


Figura 8 Layout operatori

2.4 Design procedurale

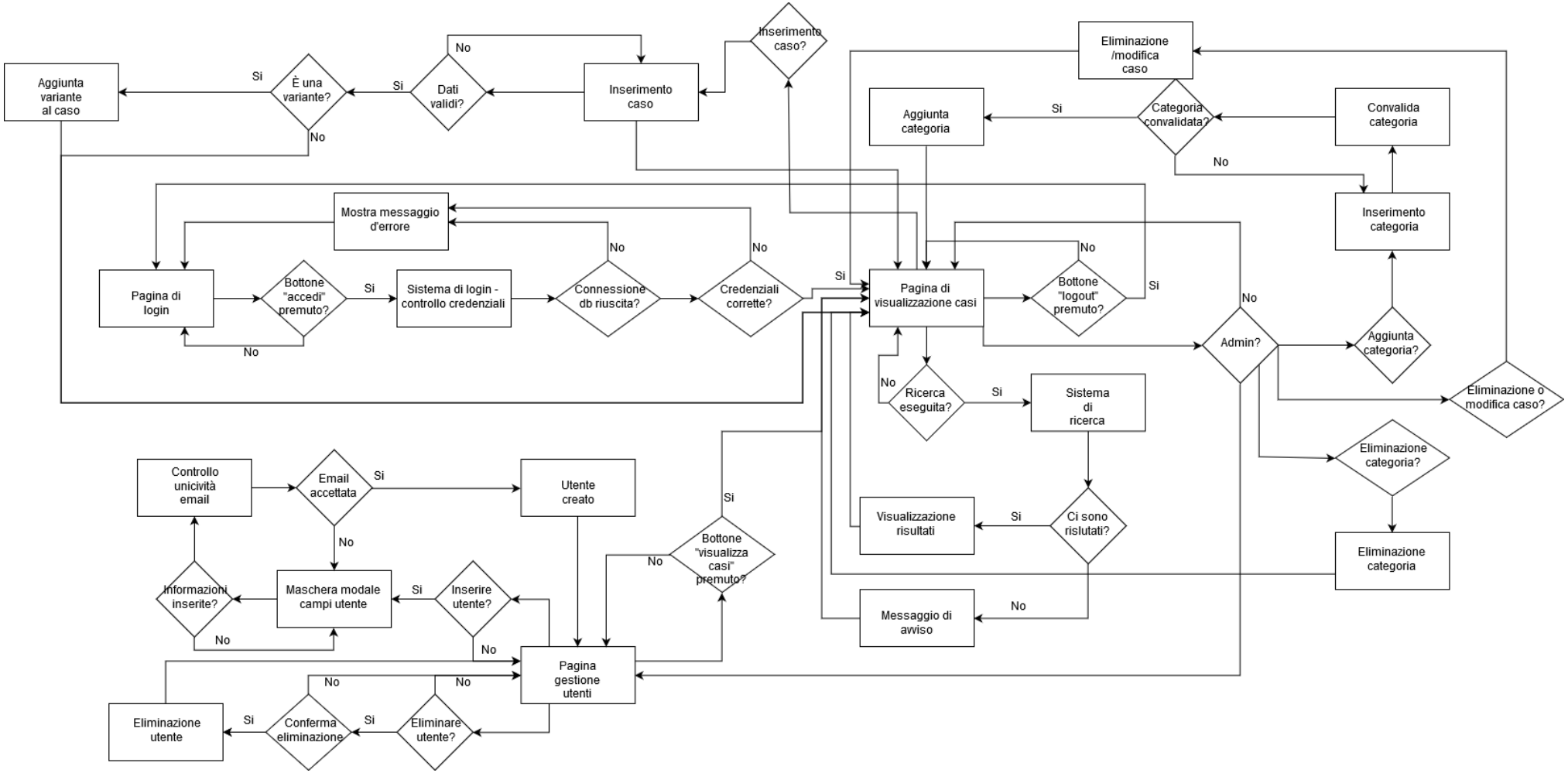


Figura 9 Diagramma procedurale

3 Implementazione

3.1 Database

Ho progettato il database ragionando a tutti gli scenari possibili e presentabili durante la navigazione all'interno dell'applicativo.

Durante l'implementazione mi sono accorto che alcune cose pianificate durante la parte di progettazione non erano ottimizzate, quindi ho dovuto effettuare delle modifiche alla struttura della banca dati.

Le modifiche che ho effettuato erano sulle colonne di alcune tabelle in cui mancavano delle condizioni, ad esempio l'attributo password non può essere nullo.

```
ALTER TABLE users ADD COLUMN password varchar(255) NOT NULL;
```

Inoltre ho aggiunto in un secondo momento che quando avviene un update di un record, in automatico viene impostato il timestamp della modifica.

```
ALTER TABLE cases CHANGE `modified_at` `modified_at` TIMESTAMP
NULL ON UPDATE CURRENT_TIMESTAMP;
```

3.2 Framework mvc

Per questo progetto ho utilizzato il framework mvc (il template creato dal docente Sartori). La struttura delle cartelle corrisponde alla seguente:

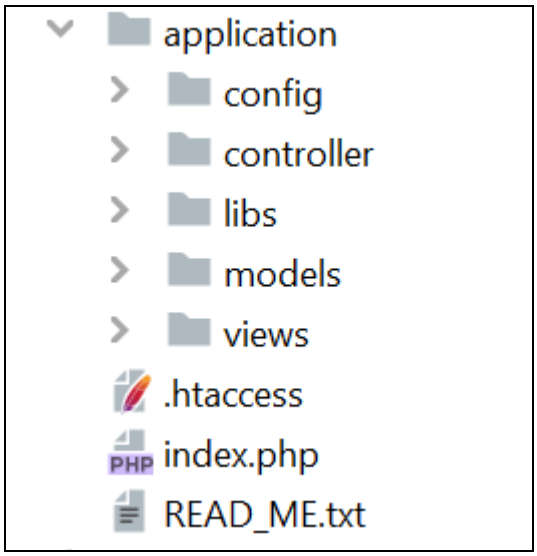


Figure 10 Struttura del progetto

Come vediamo la cartella principale application include i controller, le librerie, models, views ed alcuni file di configurazione (.htaccess e cartella config).

I controller sono delle classi che si occupano di effettuare delle operazioni (ad esempio prendere i dati da un determinato model) e richiamare delle views.

Le classi models sono delle classi che lavorano con i dati delle varie tabelle oppure che possiedono metodi per la validazione di input.

Nella cartella views invece sono presenti i layout (praticamente tutto html se non qualche parte di codice in php per rendere le pagine dinamiche).

3.2.1 Come funziona

Il funzionamento è molto semplice, nei controller bisogna inserire dei metodi pubblici che servono ad effettuare delle operazioni (ad esempio mostrare la pagina di login). Andando quindi nel browser bisogna inserire tutto il percorso del progetto fino ad arrivare al livello precedente della cartella **application**.

Esempio in questo progetto: http://localhost:8080/knowledge_base

Successivamente inserire il carattere "/" e digitare il nome del controller che si desidera utilizzare ed infine, dopo aver inserito sempre il carattere "/", inserire il nome del metodo che si vuole richiamare.

Esempio: http://localhost:8080/knowledge_base/home/index

In questo modo verrà mostrata il layout della pagina che viene chiamata all'interno del metodo **index()** del controller Home.php. Se non viene inserito il nome del metodo oppure né del metodo né del controller vengono utilizzati dei parametri di default configurati nel file application.php presente nella cartella libs.

Viene chiamato il controller di default anche nel caso che venga digitato un controller non esistente (stessa cosa vale per i metodi).

```
if (file_exists('./application/controller/' . $this->url_controller . '.php')) {

    //controllo se il metodo esiste
    if (method_exists($this->url_controller, $this->url_action)) {

    } else {
        //chiamo il metodo di default
        $this->url_controller->index();
    }
} else {

    //controller di default e metodo di default
    require './application/controller/home.php';
    $home = new Home();
    $home->index();
}
```

3.2.2 Configurazione

Per configurare in modo corretto il framework bisogna modificare il percorso nel file .htaccess a partire dalla root del webserver al livello prima della cartella application.

```
# If your app is in the root of your web folder, then leave it commented out
RewriteBase /knowledge_base
```

Infine configurare il percorso nel file config.php presente nella cartella config. Bisogna impostare il percorso inserendo l'indirizzo del server con la porta del webserver ed il percorso del progetto a partire dalla cartella root del webserver:

```
/**
 * Configurazione di : URL del progetto
 */
define('URL', 'http://localhost:8080/knowledge_base/');
```

3.2.3 Riflessione personale

Personalmente questo framework mi ha facilitato il lavoro durante lo sviluppo permettendo la separazione modulare tra i controller e le classi del models, oltre alla divisione dalle views. Mi ha reso il lavoro più semplice e la parte di astrazione modulare più intuitiva. Sono quindi soddisfatto di aver utilizzato questo framework per sviluppare l'applicativo web.

Controllers

3.2.4 Home

Nel controller home.php ci sono pochi metodi, la maggior parte di essi sono statici e vengono utilizzati dagli altri controller.

Questi sono i metodi che la classe Home contiene:

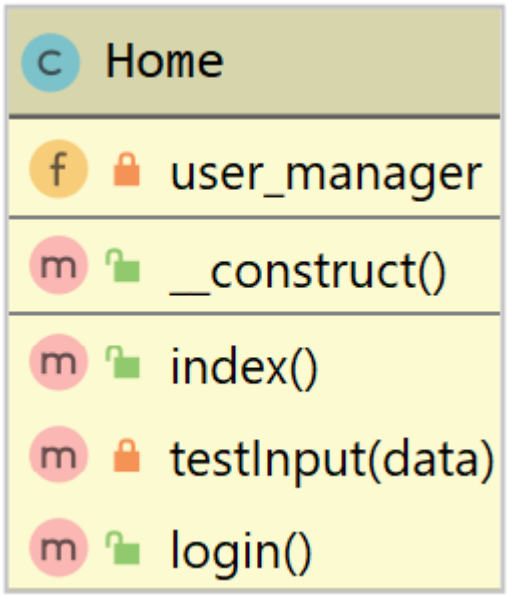


Figure 11 Classe Home

L'attributo privato **$user_manager** è un riferimento alla classe UserManager che contiene i metodi utili per gli utenti, in questo caso serve per il login.

Questo attributo viene inizializzato nel costruttore del controller in un in un blocco **try catch**. Questo perché nel costruttore della classe UserManager viene chiamato il metodo statico di connessione al database. Se questo fallisce allora l'attributo **$user_manager** sarà **null**.

In ogni metodo viene controllato se l'attributo è **null** o meno in modo da chiamare la pagina di errore di connessione al database se la connessione fallisse.

3.2.4.1 Metodo Index

Il metodo index è un semplice metodo che mostra la pagina di login al client e contiene il seguente codice.

```
if (isset($this->user_manager)) {
    require_once 'application/views/templates/head.php';
    require_once 'application/views/login/index.php';
} else {
    //redirect to no connection page
    DbError::noDatabaseConnection();
}
```

3.2.5 DbError

Il controller dbError contiene un unico metodo statico **noDatabaseConnectio()** che viene richiamato dagli altri controller quando la connessione al database fallisce.

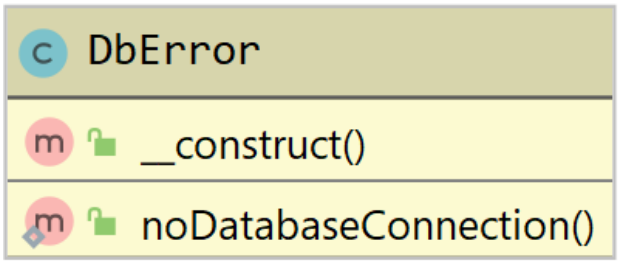


Figure 12 Classe BdError

Questo metodo non fa altro che richiamare il metodo statico **logout()** della classe UserManager.php in modo che l'utente connesso al momento del fallimento di connessione venga scollegato.

Mostra inoltre all'utente la pagina di errore di connessione al database.

```
public static function noDatabaseConnection()
{
    UserManager::logout();
    require_once 'application/views/templates/head.php';
    require_once 'application/views/errors/database_error.php';
}
```

3.2.6 Users

Il controller users.php contiene metodi utili per la gestione degli utenti. Il controller contiene i seguenti metodi.

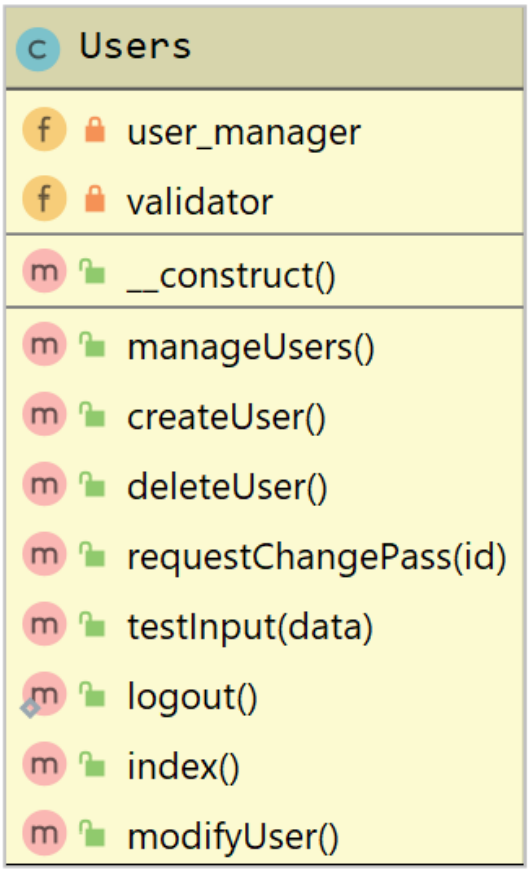


Figure 13 classe Users

Questa classe contiene un'attributo **$user_manager** che è un riferimento alla classe in cui ci sono i metodi che effettuano operazione sui dati degli utenti. **$validator** invece è un riferimento alla classe Validator.php.

3.2.6.1 Metodo manageUsers

Il metodo manageUsers serve a mostrare la pagina di gestione degli utenti. In questo metodo salvati i dati degli utenti (**$user_manager->getUsersList()**).

```
//get users list
$users = $this->user_manager->getUsersList();
```

3.2.6.2 Metodo deleteUser

Il metodo deleteUser prova ad eliminare l'utente che possiede come id, il numero identificativo passato nella variabile **$_POST['userToDelete']**. Viene utilizzato il metodo **deleteUser(** della classe UserManager.php.

```
$id = intval($this->testInput($_POST['userToDeleteId']));

//try to delete user
$this->user_manager->deleteUser($id)
```

3.2.6.3 Metodo logout

Il metodo logout non fa altro che richiamare il metodo statico **logout()** della classe UserManager.php e reindirizzare l'utente alla pagina di login.

3.2.6.4 Metodo createUser

In questo metodo vengono controllati i campi inseriti dall'utente al momento che vuole creare un utente. I campi vengono controllati tramite i metodi della classe validator.

Se i campi sono validi, utilizzando il metodo **createUser($user)** della classe UserManager.php, viene effettuato un tentativo di creazione dell'utente (non sempre va a buon fine).

```
//try to create user
$this->user_manager->createUser($user)
```

La password viene criptata con l'algoritmo **bcrypt**. Questo viene fatto con il metodo **password_hash($password, PASSWORD_DEFAULT)** in cui viene passato come primo parametro la password da criptare e secondariamente l'algoritmo da utilizzare.

```
//hash password
$password = password_hash($password, PASSWORD_DEFAULT);
```

3.2.6.5 Metodi requestChangePass e testInput

Il metodo **requestChangePass()** è implementato nel codice ma la funzione non è abilitata nell'applicativo. Questo metodo richiama un metodo sempre della classe di gestione degli utenti che cambia il valore dell'attributo change_pass del database (vedi implementazioni future).

Il metodo testInput corrisponde esattamente al metodo del controller home.php.

3.2.6.6 Metodo index

Questo metodo non fa altro che chiamare il metodo **manageUsers()**. Questo metodo viene richiamato se l'utente digita nell'url un metodo che non è presente nel controller.

3.2.6.7 Metodo modifyUser

Questo metodo viene invocato quando si tenta di modificare un utente esistente (dalla pagina di gestione utenti alla quale solo gli admin hanno accesso).

Questo metodo corrisponde al metodo **createUser()** con la particolarità che controlla che la password sia vuota o meno:

- Password vuota → utente non vuole modificare la password corrente
- Password inserita → vengono effettuati i controlli di complessità della password

Se la password è settata viene effettuato l'hash sempre con l'algoritmo bcript.

```
//set empty password
$password = "";

//check if there is a password
if (isset($_POST['modified_password']) && !empty($_POST['modified_password'])) {

    $password = $this->testInput($ POST['modified password']);

    //check if the password is complex enough
    if (!PasswordManager::checkStrength($password)) {
        MessageManager::setErrorMsg("La password non rispetta le condizioni di
sicurezza");
        MessageManager::printErrorMsg();
        $this->manageUsers();
        exit();
    }

    //hash password
    $password = password_hash($password, PASSWORD_DEFAULT);
}
```

Come avviene nel metodo di creazione viene inizializzato un oggetto User a cui si passano i parametri dell'utente modificato che viene passato al metodo **modifyUser()** della classe UserManager.php.

```
//try to update user
if ($this->user_manager->modifyUser($user, $id)) {
```

3.2.7 ResearchCases

In questo controller vi sono tutti i metodi utili per svolgere le funzioni implementate nella pagina ricerca_casi.php. Questo controller contiene i seguenti metodi.

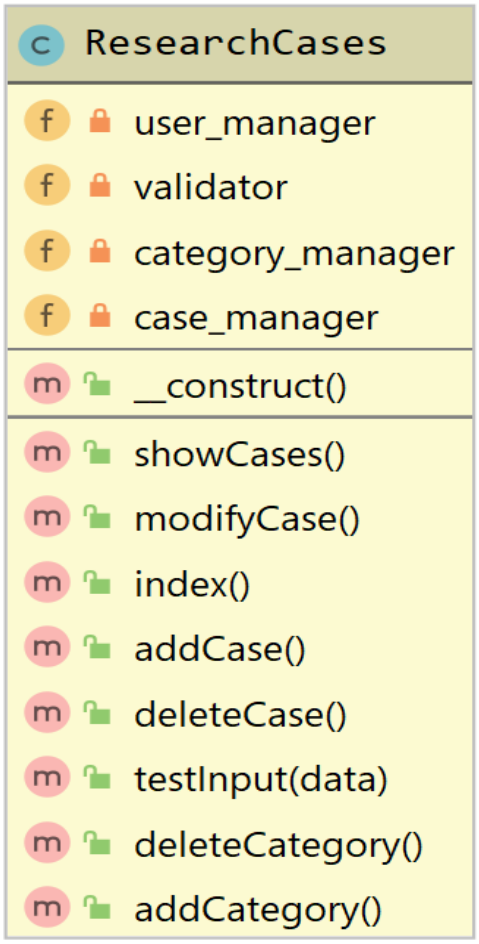


Figure 14 Classe ResearchCases

Questo controller possiede degli attributi che sono dei riferimenti alle classi che servono per la gestione dei casi, utenti, categorie e validator.

3.2.7.1 Metodo showCases

Questo metodo viene richiamato quando l'utente vuole visualizzare la pagina ricerca_casi.php. Questo metodo controlla se le variabili per i filtri sono applicati e successivamente, se i filtri sono attivi, crea e salva all'interno delle variabili di sessione i vari filtri. Una volta fatto ciò richiama il metodo che ritorna i risultati della query effettuata nel metodo **getCases()** della classe CasesManager.php.

```
//get cases
$cases = $this->case_manager->getCases();
```

In base ai valori inseriti nei campi di ricerca vengono settate delle variabili di sessione che verranno utilizzate successivamente nella classe CaseManager.php. La variabile $_POST["order_results"] contiene un valore intero che corrisponde al tipo di ordinamento:

- Se il valore è 0 → ordinamento dal caso più recente
- Se il valore è 1 → ordinamento dal caso con più ripresentazioni
- Se il valore è 2 → ordinamento dal caso meno recente

```
if (isset($_POST['order_results']) && intval($_POST['order_results']) == 1) {

    //order by most represent
    $_SESSION['order_results'] = 1;
} else if (isset($_POST['order_results']) && intval($_POST['order_results']) ==
2) {

    //order by less recent
    $_SESSION['order_results'] = 2;
} else {
    //order by most recent
    $_SESSION['order_results'] = 0;
}
```

Successivamente, utilizzando il metodo **getAllCases**, salvo in una variabile tutti i casi (quindi senza filtri) per stampare le opzioni all'interno del menu dropdown durante la modifica di un caso.

```
$all_cases = $this->case_manager->getAllCases();
```

Infine, l'utente, viene reindirizzato l'utente alla pagina ricerca_casi.php.

3.2.7.2 Metodi deleteCategory e deleteCase

I due metodi vengono chiamati quando un utente con privilegi da admin, per eliminare una categoria o rispettivamente un caso.

Nel metodo **deleteCategory()** viene chiamato il metodo **deleteCategory($id)** della classe CategoryManager.php.

```
$category_id = $this->testInput($_POST['delete_category_id']);

//try to delete category
$this->category_manager->deleteCategory($category_id);
```

Nel metodo **deleteCase** viene svolto lo stesso procedimento per l'eliminazione di un caso.

```
$id = intval($this->testInput($_POST['caseToDeleteId']));

//try to delete the case
$this->case_manager->setDeletedCase($id));
```

3.2.7.3 Metodo index

Questo metodo non fa altro che chiamare il metodo **showCases()**. Questo metodo viene richiamato se l'utente digita nell'url un metodo che non è presente nel controller.

3.2.7.4 Metodi addCase e addCategory

Il procedimento è molto simile a quello per l'aggiunta di un nuovo utente. In entrambi i metodi vengono controllati che le rispettive variabili **$_POST** siano settate e tramite la classe Validator.php che i loro valori rispettino la sintassi corretta.

Aggiunta categoria:

```
$new_category = $this->testInput($_POST['new_category']);

$this->validator->validateTextInput($new_category, 1, 50);

//add new category
if ($this->category_manager->addCategory($new_category)
```

Stesso procedimento per l'aggiunta di un caso, ogni campo viene validato.

```
//create DocCase object
$case = new DocCase($title, $category, $variant, $description);

$this->case_manager->addCase($case);
```

3.2.7.5 Metodo modifyCase

Questo metodo viene chiamato quando l'utente decide di salvare delle modifiche apportate ad un caso. Una volta effettuati i controlli sulla sintassi dei valori inseriti, utilizza un metodo della classe CaseManager.php per modificare il caso richiesto.

```
//create DocCase object
$case = new DocCase($title, $category, $variant, $description);

$this->case_manager->modifyCase($case, $id);
```

3.3 Models

All'interno della cartella models ci sono tutte le classi che interagiscono con i dati del database (effettuano query), classi che modellizzano gli oggetti (caso, utente), classi per la validazione dati, per la connessione alla banca dati.

I metodi di eliminazione, modifica e aggiunta dati **ritornano un valore booleano** per sapere se l'operazione è andata a buon fine o meno.

3.3.1 DbManager

Questa classe possiede un unico metodo statico che serve ad effettuare la connessione al database.

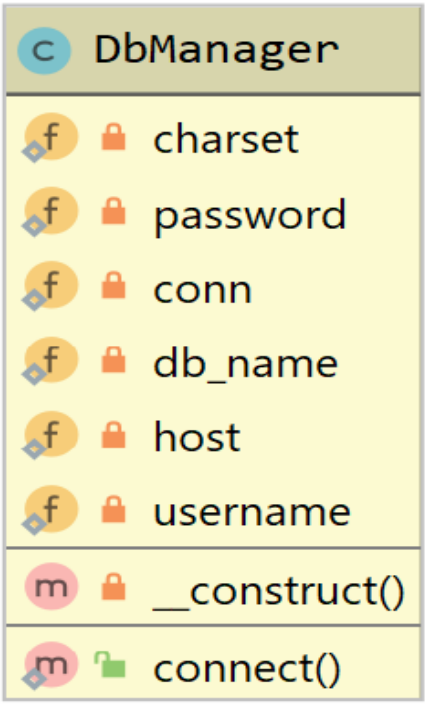


Figure 15 Classe DbManager

Questo metodo viene utilizzato da tutte le classi, controller che devono connettersi o controllare la connessione al database.

```
public static function connect()
{
    if (!self::$conn) {

        self::$conn = new PDO("mysql:host=" . self::$host . ";dbname=" .
self::$db_name . ";charset=" . self::$charset, self::$username,
self::$password);

        // set the PDO error mode to exception
        self::$conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
        return self::$conn;
    }
    return self::$conn;
}
```

Nelle altre classi per controllare che la connessione sia riuscita creo una variabile e le assegno il valore ritornato dal metodo **connect**(). Successivamente nei vari metodi controllo che il suo valore non sia **null**. Se la variabile risulta **null** significa che la connessione è fallita.

```
private $conn;

public function   construct()
{
    try {
        $this->conn = DbManager::connect();
    } catch (PDOException $ex) {
        throw $ex;
    }
}
```

3.3.2 Validator

La classe Validator.php contiene un metodo **validateTextInput()** che controlla che il parametro di testo passato il metodo contenga un numero di caratteri che sia compreso tra il numero minimo e massimo desiderato.

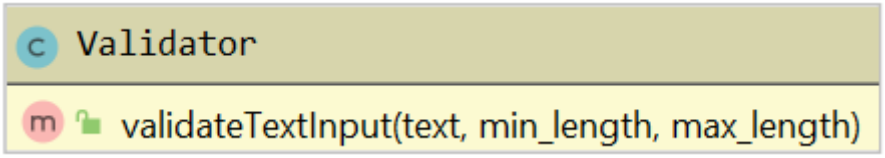


Figure 16 Classe Validator

Questo metodo controlla quindi che il testo non contenga più o meno caratteri del consentito.

```
public function validateTextInput($text, $min_length, $max_length){
    return (strlen($text) >= $min_length && strlen($text) <= $max_length);
}
```

La variabile **$text** corrisponde al testo da controllare, **$min** il numero minimo di caratteri e **$max** il numero Massimo di caratteri consentiti.

3.3.3 PasswordManager

La classe password manager contiene due metodi statici utili per la gestione delle password.

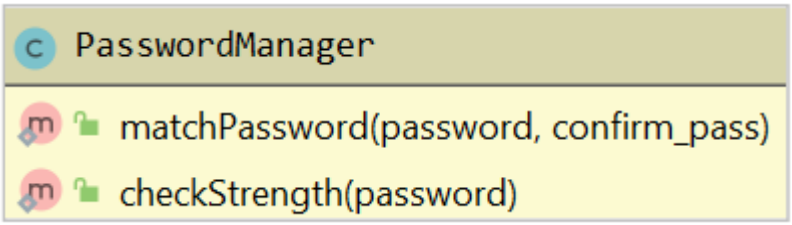


Figure 17 Classe PasswordManager

Entrambi I metodi vengono utilizzati all'interno del controller users.php nel metodo **createUser()**.

3.3.3.1 Metodo matchPassword

Questo metodo confronta che la password inserita dall'utente corrisponda alla password di conferma, in modo da evitare che l'utente sbagli ad inserire la prima password.

Bisogna passare come parametri la password e la password di conferma.

```
public static function matchPassword($password, $confirm_pass)
{
    return $password === $confirm_pass;
}
```

3.3.3.2 Metodo checkStrength

Nel metodo **checkStrength($password)**, tramite le regular expression, viene controllato che la password inserita dall'utente rispetti una sintassi di sicurezza minima:

- Almeno 8 caratteri
- Un carattere maiuscolo
- Massimo 50 caratteri
- Almeno un numero

```
public static function checkStrength($password)
{

    //length 8, 1 uppercase, at least 1 digit, max length 50 characters
    $pattern = '/^(?=.*[0-9])(?=.*[A-Z]).{8,50}$/';

    return preg_match($pattern, $password);
}
```

Si deve passare come parametro la password da controllare.

3.3.4 DocCase

Questa classe modellizza l'oggetto "caso". Questa classe contiene i metodi **getter** dei vari attributi che vengono settati nel costruttore.

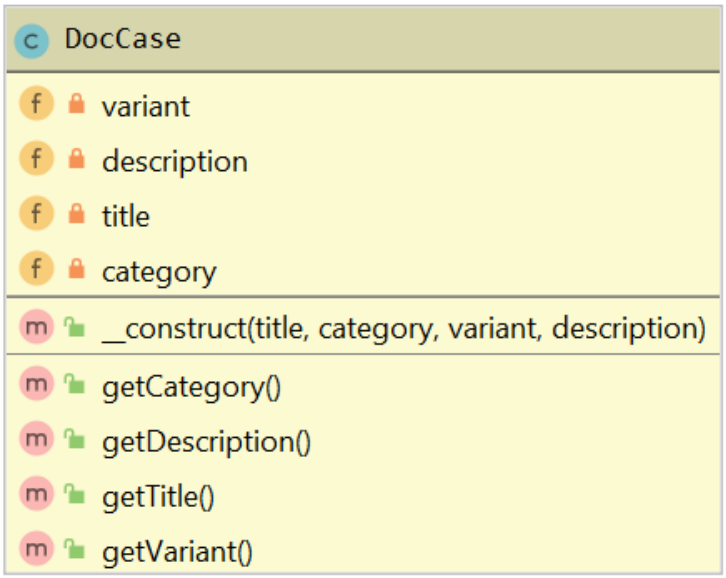


Figure 18 Classe DocCase

Questa classe la utilizzo all'interno del metodo **addCase()** del controller researchCases.php, in cui passo alla classe CaseManager.php l'oggetto **CaseDoc** per evitare di passare tutti i parametri a mano e non farli gestire a quest'ultima.

3.3.4.1 Metodo __construct

Questo metodo richiede come parametri il titolo del caso, l'id della categoria alla quale appartiene, l'id della variante (**null** se non è una variante di nessun caso) e la descrizione del caso.

```
public function __construct($title, $category, $variant, $description)
{
    $this->title = $title;
    $this->category = $category;
    $this->variant = $variant;
    $this->description = $description;
}
```

3.3.4.2 Metodi getter

Questi metodi non fanno altro che ritornare il valore dei rispettivi attributi. Qui sotto è illustrato un esempio di metodo getter: **getTitle()**

```
public function getTitle()
{
    return $this->title;
}
```

3.3.5 User

Questa classe ha la stessa funzione della classe DocCase.php ma l'oggetto modellizzato in questo caso è "l'utente". Anche in questo caso sono presenti solo i metodi **getter** ed il costruttore.

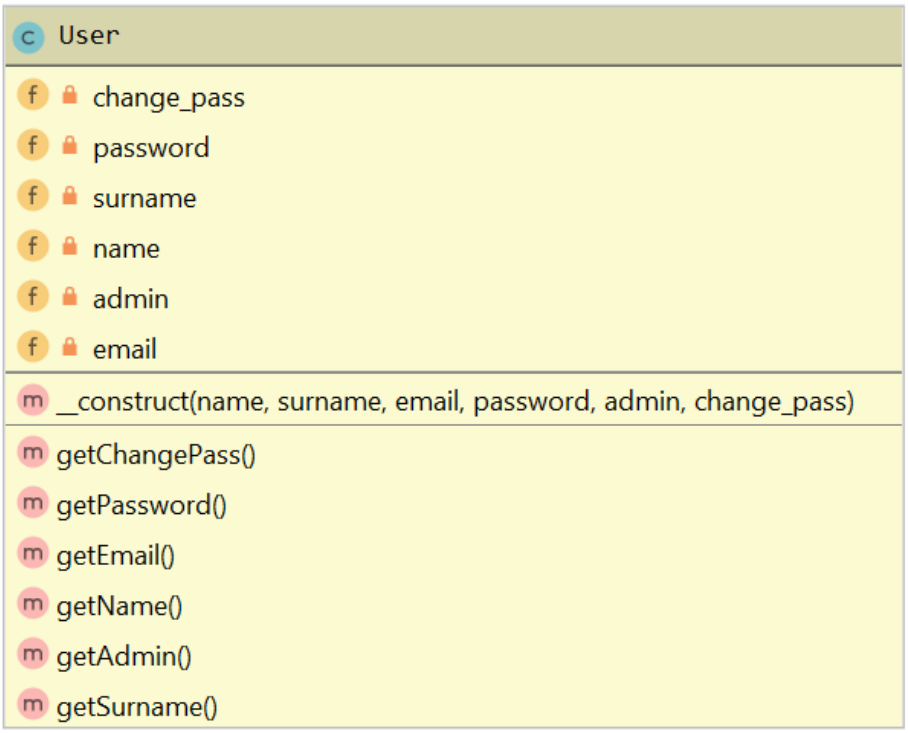


Figure 19 Classe User

Questa classe viene utilizzata dal controller users.php nel metodo di creazione dell'utente (**createUser($user)**).

3.3.5.1 Metodo __construct

Il metodo **__construct()** non fa altro che assegnare i valore ricevuti come parametro ai relativi attributi. Richiede il nome dell'utente, il cognome, la sua email, la password, se è admin o meno (**1** corrisponde ad utente admin) e se è stato richiesto un cambio di password (implementazione futura, valore di default **0**).

```
public function __construct(string $name, string $surname, string $email, string
$password, $admin, $change_pass )
{
    $this->name = $name;
    $this->surname = $surname;
    $this->email = $email;
    $this->password = $password;
    $this->admin = $admin;
    $this->change_pass = $change_pass;
}
```

3.3.6 CategoryManager

La classe CategoryManager.php è una classe che serve a lavorare con i dati presenti nel database relativi alla categoria. La classe possiede i seguenti metodi:

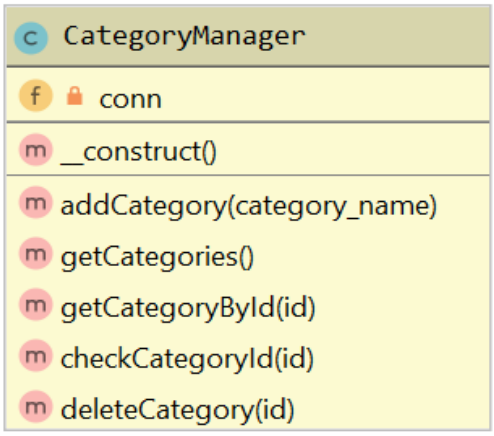


Figure 20 Classe CategoryManager

3.3.6.1 Metodi getCategories e getCategoryById

Questi due metodi sono utilizzati nel controller researchCase.php, svolgono una funzione molte simile con una differenza.

Il metodo **getCategories()** ritorna tutte le categorie presenti all'interno del sistema.

```
public function getCategories(){
    try {
        //prepare query
        $prepared_query = $this->conn->prepare("SELECT * FROM categories");
        $prepared_query->execute();
        return $prepared_query->fetchAll();

    } catch (PDOException $ex) {
    }
}
```

Il metodo **getCategoryById($id)** ritorna la categoria desiderato passando come parametro l'id della categoria.

```
public function getCategoryById($id){
    try {
        //prepare query
        $prepared_query = $this->conn->prepare("SELECT name FROM categories
WHERE id = :id");
        $prepared_query->bindParam(':id', $id, PDO::PARAM_INT);
        $prepared_query->execute();

        $res = $prepared_query->fetch();
        return $res[0];

    } catch (PDOException $ex) {
    }
}
```

3.3.6.2 Metodi addCategory e deleteCategory

Il metodo **addCategory($category_name)** è un metodo che viene utilizzato quando l'utente prova a creare una nuova categoria e richiede il nome della categoria come parametro.

In questo caso non ho creato la classe Category.php perché la categoria richiede solo il nome.

```
public function addCategory($category_name){
    try {
        //prepare query
        $prepared_query = $this->conn->prepare("INSERT INTO categories(name)
VALUES (:category_name)");
        $prepared_query->bindParam(':category_name', $category_name,
PDO::PARAM_STR);
        $prepared_query->execute();

        return true;
    } catch (PDOException $ex) {
        Home::setErrorMsg("La categoria esiste già");
        return false;
    }
}
```

Il metodo **deleteCategory($id)** serve ad eliminare una categoria dal database. Come parametro viene richiesto l'id della categoria.

```
public function deleteCategory($id){
    try {
        //prepare query
        $prepared_query = $this->conn->prepare("DELETE FROM categories WHERE id
= :id");
        $prepared_query->bindParam(':id', $id, PDO::PARAM_INT);
        $prepared_query->execute();

        return true;
    } catch (PDOException $ex) {
        echo $ex;
        Home::setErrorMsg("Impossibile eliminare questa categoria");
        return false;
    }
}
```

3.3.6.3 Metodo checkCategory

Questo metodo controlla e ritorna se esiste una categoria che possiede come id l'identificatore passato come parametro.

```
public function checkCategoryId($id)
{
    try {
        $prepared_query = $this->conn->prepare("SELECT * FROM CATEGORIES WHERE
id = :id");
        $prepared_query->bindParam(":id", $id, PDO::PARAM_INT);
        $prepared_query->execute();
        $res = $prepared_query->fetchAll(PDO::FETCH_ASSOC);

        return sizeof($res) > 0;

    } catch (PDOException $ex) {
    }
}
```

Questo metodo serve per evitare che l'utente inserisca una categoria (id della categoria), durante la creazione o modifica di un caso, che non è presente nella banca dati.

3.3.7 MessageManager

Questa classe contiene dei metodi che vengono utilizzati per mostrare all'utente dei messaggi di errore o di successo.

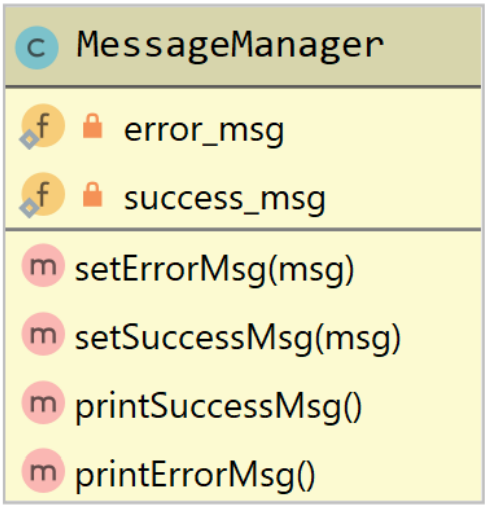


Figure 21 Classe MessageManager

3.3.7.1 Metodi setErrorMsg e setSuccessMsg

Questi due metodi contengono lo stesso codice con l'unica differenza dell'azione che svolgono. Il metodo **setErrorMsg($msg)** imposta il testo passato come parametro alla variabile statica **$error_msg** mentre **setSuccessMsg($msg)** alla variabile statica **$success_msg**.

3.3.7.2 Metodi printErrorMsg e printSuccessMsg

Questi due metodi non fanno altro che stampare, tramite degli alert, creati utilizzando mdbootstrap, le stringhe salvate all'interno delle variabili **$error_msg** e **$success_msg**.

Figure 22 Messaggio di errore

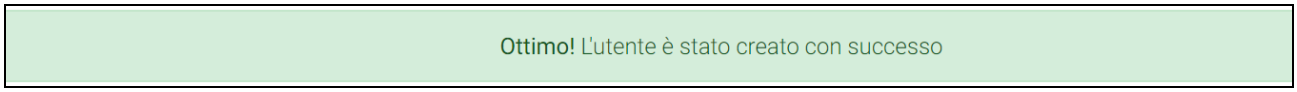


Figure 23 Messaggio di successo

3.3.8 CaseManager

Questa classe viene utilizzata per effettuare tutte le operazioni sui dati relativi ai casi di documentazione tecnica. La classe presenta i seguenti metodi.

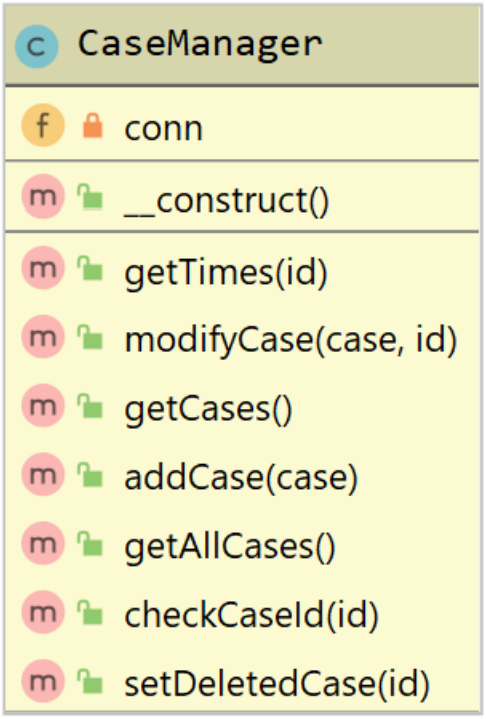


Figure 24 Classe CaseManager

3.3.8.1 Metodo getCases()

Questo metodo ritorna i casi applicando i filtri. Nel controller researchCases.php nel metodo **showCases()** vengono settate le variabili di sessione utilizzati per i filtri.

```
//set session variables value
(isset($_SESSION['text_filter'])) ? $text_filter = $_SESSION['text_filter'] :
$text_filter = "";
(isset($_SESSION['date_filter'])) ? $date_filter = $_SESSION['date_filter'] :
$date_filter = "";

//if value == 0 -> all categories
(isset($_SESSION['category_filter']) && intval($_SESSION['category_filter']) !=
0) ? $category_filter = $_SESSION['category_filter'] : $category_filter = "";
```

Se le variabili di sessione non sono state inizializzate, la query che viene eseguita successivamente non applica nessun filtro di ricerca perché le variabili filtri contengono il valore "**%%**" che ritorna tutti i risultati. Altrimenti se contiene ad esempio del testo viene effettuata la query in cui si cercano tutti i casi che contengano quel testo all'interno del **titolo** o della **descrizione**.

```
$id = $text_filter;
$text_filter = "%" . $text_filter . "%";
$date_filter = "%" . $date_filter . "%";
$category_filter = "%" . $category_filter . "%";

if (intval($_SESSION['order_results']) == 0 ||
    intval($_SESSION['order_results']) == 2) {
```

Sul campo **id** e **variante** viene fatta la query precisa, quindi il numero deve corrispondere esattamente a quello richiesto al contrario del **titolo** e la **descrizione** che devono contenere il numero. Questa è la query per quanto riguarda l'ordinamento per data (dal più recente o viceversa).

```
//order by date
$sql = "SELECT * FROM cases WHERE DELETED = 0 AND
                  (description LIKE :text_filter
                  OR id LIKE :id
                  OR title LIKE :text_filter
                  OR variant LIKE :id)
                  AND (created_at LIKE :date_filter)
                  AND (category_id LIKE :category_id" .
                  $include_null_category_id .
                  "order by created_at " . $order;
```

Se il tipo di ordinamento corrisponde a quello dal caso più ricorrente a quello meno ricorrente viene eseguita una query leggermente più complessa che effettua una join sulla stessa tabella in modo da ritornare i casi che sono stati ripresentati più volte in ordine ascendente.

```
$sql = "SELECT c.* FROM cases c
        INNER JOIN cases a ON
        c.id = a.variant
        where a.deleted = 0 AND a.variant IS NOT NULL
        AND (c.description LIKE :text_filter
            OR c.id LIKE :id
            OR c.title LIKE :text_filter
            OR c.variant LIKE :id)
        AND (c.created_at LIKE :date_filter)
        AND (c.category_id LIKE :category_id" . $include_null_category_id .
    " group by a.variant
        order by count(a.variant) desc;";
```

Successivamente, indipendentemente dalla query, viene chiamato il metodo della classe PDO **prepare()** per evitare sql injection. Infine viene ritornato il risultato della query al relativo controller.

```
$prepared_query = $this->conn->prepare($sql);

//bind params
$prepared_query->bindParam(':text_filter', $text_filter, PDO::PARAM_STR);
$prepared_query->bindParam(':id', $id, PDO::PARAM_STR);
$prepared_query->bindParam(':date_filter', $date_filter, PDO::PARAM_STR);
$prepared_query->bindParam(':category_id', $category_filter, PDO::PARAM_STR);

$prepared_query->execute();

return $prepared_query->fetchAll(PDO::FETCH_ASSOC);
```

3.3.8.2 Metodo getAllCases

Questo metodo ritorna tutti i casi senza applicare filtri.

```
$prepared_query = $this->conn->prepare("SELECT * FROM CASES WHERE deleted = 0");
$prepared_query->execute();
$res = $prepared_query->fetchAll(PDO::FETCH_ASSOC);

return $res;
```

3.3.8.3 Metodo checkCaseId

Questo metodo ritorna un valore booleano che indica se esiste o meno un caso che possiede come id il valore passato come parametro del metodo.

```
$prepared_query = $this->conn->prepare("SELECT * FROM CASES WHERE deleted = 0
AND ID = :id");
$prepared_query->bindParam(":id", $id, PDO::PARAM_INT);
$prepared query->execute();
$res = $prepared_query->fetchAll(PDO::FETCH_ASSOC);

return sizeof($res) > 0;
```

3.3.8.4 Metodo addCase e setDeleteCase

Il metodo **addCase($case)** richiede come parametro un oggetto di tipo DocCase relativo al nuovo caso. Questo metodo prova ad inserire nella tabella cases il nuovo record. Per ottenere le informazioni vengono utilizzati i metodi della classe DocCase.php.

```
//get values
$title = $case->getTitle();
$description = $case->getDescription();
$category_id = $case->getCategory();
$variant = $case->getVariant();

//prepare query
$prepared_query = $this->conn->prepare($sql);
```

Il metodo **setDeletedCase($id)** serve a settare l'attributo **deleted** della tabella **cases** del caso con id passato come parametro ad 1. In questo modo il caso non viene realmente eliminato, ma non viene più mostrato agli utenti.

Prima di tutto bisogna settare a **null** i campi **variant** della tabella dei casi che possiedono come variante il caso che si desidera eliminare.

```
//get cases that have the variant id of the deleted case
$variant_query = $this->conn->prepare("SELECT * FROM CASES WHERE variant =
:id");
$variant_query->bindParam(':id', $id, PDO::PARAM_INT);
$variant_query->execute();

$results = $variant_query->fetchAll(PDO::FETCH_ASSOC);

//set null variant field
foreach ($results as $result) {
    $prepared_query = $this->conn->prepare("UPDATE CASES set variant = null
where id = :id");
    $prepared_query->bindParam(':id', $result['id'], PDO::PARAM_INT);
    $prepared_query->execute();
}
```

Successivamente si setta a 1 il campo **deleted** del caso passato come parametro.

```
//prepare query
$prepared_query = $this->conn->prepare("UPDATE CASES SET deleted = 1 WHERE ID =
:id");
$prepared_query->bindParam(':id', $id, PDO::PARAM_INT);
$prepared_query->execute();
//prepare query
$prepared_query = $this->conn->prepare("UPDATE CASES SET deleted = 1 WHERE ID =
:id");
$prepared_query->bindParam(':id', $id, PDO::PARAM_INT);
$prepared_query->execute();
```

3.3.8.5 Metodo getTimes

Questo metodo ritorna il numero di volte che il caso che possiede l'id passato come parametro del metodo è stato ripresentato come variante. Questo metodo è stato ideato per evitare di fare ogni volta una join della tabella per ottenere il risultato, così da migliorare le prestazioni.

```
//get times
$prepared_query = $this->conn->prepare("SELECT count(*) FROM cases WHERE variant
= :id and deleted = 0");
$prepared_query->bindParam(':id', $id, PDO::PARAM_INT);
$prepared_query->execute();

$res = $prepared_query->fetch();

return intval($res[0]);
```

3.3.8.6 Metodo modifyCase

Questo metodo richiede come parametro l'id del caso che si desidera modificare. Mentre il parametro **$case** che è l'oggetto DocCase che contiene il caso modificato. Come nel metodo **addCase()**, utilizzando i metodi della classe DocCase-php si ottiene il titolo del caso, la descrizione, variante e categoria.

Una volta fatto ciò bisogna aggiornare il caso esistente con i nuovi dati.

```
$sql = "UPDATE cases SET title = :title, description = :description, category_id
= :category_id, variant = :variant WHERE ID = :id;";

//get values
$title = $case->getTitle();
$description = $case->getDescription();
$category_id = $case->getCategory();
$variant = $case->getVariant();

…

//prepare query
$prepared_query = $this->conn->prepare($sql);

…

$prepared_query->execute();
```

3.3.9 UserManager

La classe UserManager.php ha la stessa funzione della classe CaseManager.php ed include alcuni metodi aggiuntivi. La classe possiede la variabile di connessione al database, presente in tutte le classi che lavorano utilizzando la banca dati **$conn** ed i seguenti metodi.

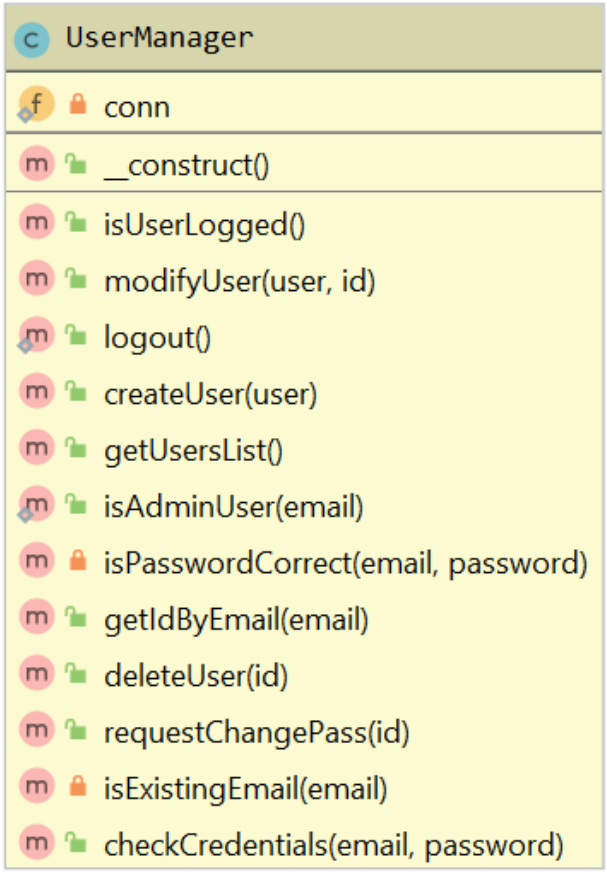


Figure 24 Classe UserManager

3.3.9.1 Metodo getUsersList

Questo metodo non fa altro che ritornare la lista di tutti gli utenti. Viene utilizzato nel controller users.php per stampare successivamente gli utenti in una lista nella pagina di gestione utenti.

```
//get the password from database
$prepared query = self::$conn->prepare("SELECT * FROM users");
$prepared_query->execute();

$users = $prepared_query->fetchAll();

return $users;
```

3.3.9.2 Metodo checkCredentials

Questo metodo controlla se, utilizzando il metodo privato **isExistingEmail($email)** che ritorna se l'email passata come parametro esiste all'interno del database e successivamente, se l'email esiste, il metodo **isPasswordCorrect($email, $password)**, le credenziali utilizzate durante il login sono corrette o meno.

```
if ($this->isExistingEmail($email)) {

    //check if the password is correct
    return $this->isPasswordCorrect($email, $password);

}

return false;
```

3.3.9.3 Metodo isUserLogged

Questo metodo controlla se l'utente è già loggato, per evitare che ad ogni azione vengano richieste le credenziali di accesso.

Per fare ciò controlla se nelle variabili di sessioni sono presenti delle credenziali valide.

```
//check if the variables are initialized
if (isset($_SESSION['email']) && isset($_SESSION['password'])) {
    return $this->checkCredentials($_SESSION['email'], $_SESSION['password']);
}
return false;
```

3.3.9.4 Metodo logout

Questo metodo statico viene utilizzato dai controller per eliminare le variabili di sessione, così da cancellare le credenziali. In questo modo per accedere alle pagine bisogna effettuare il login.

```
public static function logout()
{
    session_destroy();
}
```

3.3.9.5 Metodi isExistingEmail e isPasswordCorrect

Il metodo **isExistingEmail($email)** controlla e ritorna, tramite valore booleano, se l'email esiste all'interno della banca dati.

```
//get the number of rows that contain the user email (MAX 1)
$prepared_query = self::$conn->prepare("SELECT count(*) FROM users WHERE email =
:email");
$prepared_query->bindParam(':email', $email, PDO::PARAM_STR);
$prepared_query->execute();
$res = $prepared_query->fetch();

//check if the row count is 1
return intval($res[0]) == 1;
```

Il metodo **isPasswordCorrect($email, $password)** ritorna se la password relativa all'email passata come parametro al metodo è corretta o meno.

```
//get the password from database
$prepared_query = self::$conn->prepare("SELECT password FROM users WHERE email =
:email");
$prepared_query->bindParam(':email', $email, PDO::PARAM_STR);
$prepared_query->execute();
$res = $prepared_query->fetch();

$hashed_password = $res[0];

//check if the inserted password is correct
return password_verify($password, $hashed_password);
```

3.3.9.6 Metodo isAdminUser

Questo metodo statico ritorna se l'utente, identificato dall'email passata come parametro **$email**, è admin o meno:

- Se ritorna 0 → l'utente non è admin
- Se ritorna 1 → l'utente è admin

```
//get is_admin value
$prepared_query = self::$conn->prepare("SELECT is_admin FROM users WHERE email =
:email");
$prepared_query->bindParam(':email', $email, PDO::PARAM_STR);
$prepared_query->execute();

//get result
$res = $prepared_query->fetch();
$is_admin = $res[0];

return intval($is_admin);
```

3.3.9.7 Metodo createUser

Questo metodo prova a creare l'utente utilizzando l'oggetto User passato come parametro.

Come prima cosa controlla con il metodo **isExistingEmail($email)** che l'email non sia già utilizzata da qualche utente. Successivamente tramite i metodi della classe User.php vengono estratti i dati dell'utente e inseriti nella query.

```
if (!$this->isExistingEmail($user->getEmail())) {
. //prepare query
$prepared_query = self::$conn->prepare("INSERT INTO USERS (name, surname, email,
password, is_admin, change_pass) VALUES(:name, :surname, :email, :password,
:is_admin, :change_pass)");

//get params
$name = $user->getName();
$surname = $user->getSurname();
$email = $user->getEmail();
$password = $user->getPassword();
$is_admin = $user->getAdmin();
$change_pass = $user->getChangePass();
…

$prepared_query->execute();

return true;
```

3.3.9.8 Metodo getEmailById

Questo metodo richiede come parametro l'id dell'utente, e tramite quest'ultimo ritorna la relativa email.

```
//prepare query
$prepared_query = self::$conn->prepare("SELECT id FROM USERS WHERE EMAIL =
:email");
$prepared_query->bindParam(':email', $email, PDO::PARAM_STR);
$prepared_query->execute();

$res = $prepared_query->fetch();
return $res[0];
```

3.3.9.9 Metodo deleteUser

Questo metodo prova ad eliminare l'utente utilizzando l'id ricevuto come parametro. Se l'id non esiste l'operazione non fallisce ma nessun record viene eliminato.

```
//prepare query
$prepared_query = self::$conn->prepare("DELETE FROM USERS WHERE ID = :id");
$prepared_query->bindParam(':id', $id, PDO::PARAM_INT);
$prepared_query->execute();

return true;
```

3.3.9.10 Metodo requestChangePass

Questo metodo è stato creato ma al momento non è implementato ed utilizzato da nessuna classe (vedi capitolo implementazioni future). Questo metodo non fa altro che settare il valore dell'attributo **change_pass** della tabella utenti ad 1.

```
//prepare query
$prepared_query = self::$conn->prepare("UPDATE USERS SET change_pass = 1 WHERE
id = :id");
$prepared_query->bindParam(':id', $id, PDO::PARAM_INT);
$prepared_query->execute();
```

Questo metodo richiede come parametro l'id dell'utente che ha richiesto il cambio della password.

3.3.9.11 Metodo modifyUser

Questo metodo si occupa della modifica di un utente già esistente all'interno della banca dati. Come parametri riceve l'oggetto utente, che contiene i valori da aggiornare, e l'id dell'utente da modificare.

Come prima cosa viene controllato che l'email passata corrisponda a quella già registrata, in tal caso si evita di fare una modifica dell'email e viene quindi omessa nella query. In questo metodo la query viene costruita a tappe. Se l'email non corrisponde viene controllato che non la possieda già un altro utente.

```
//check if the email is the current user email
if ($this->getIdByEmail($user->getEmail()) === intval($id)){
    $sql .= "UPDATE USERS set name = :name, surname = :surname, is_admin =
:is_admin, change_pass = :change_pass";

} else {

    //check if the email is already used
    if(!$this->isExistingEmail($user->getEmail())){
        $change email = true;
        $sql .= "UPDATE USERS set name = :name, surname = :surname, email =
:email, is_admin = :is_admin, change_pass = :change_pass";
    }
}
```

Successivamente viene controllato che la password non sia vuota, in modo da non inserirla nella query se l'utente non ha richiesto un cambio password.

```
//check if update password
if (!empty($password)) {
    $sql .= ", password  = :password";
}
```

Infine viene inserito aggiunta la condizione nella query per identificare l'utente e successivamente, dopo aver effettuato il controllo sui vari valori, viene eseguita l'operazione.

```
$sql .= " WHERE ID = :id";

//prepare query
$prepared_query = self::$conn->prepare($sql);
…
$prepared_query->execute();
```

3.4 Pagine

Per quanto riguarda le pagine ho utilizzato il framework Material Design Bootstrap. Questo framework mette a disposizione molte classi css che facilitano, oltre a rendere il risultato più presentabile, e velocizzano il lavoro di creazioni delle interfacce.

3.4.1 Pagina di login

Per la pagina di login ho scelta una struttura molto semplice che rispecchia il design delle interfacce che ho progettato.

Vi è una maschera centrata nello schermo con all'interno i campi in cui inserire la propria email e la password.

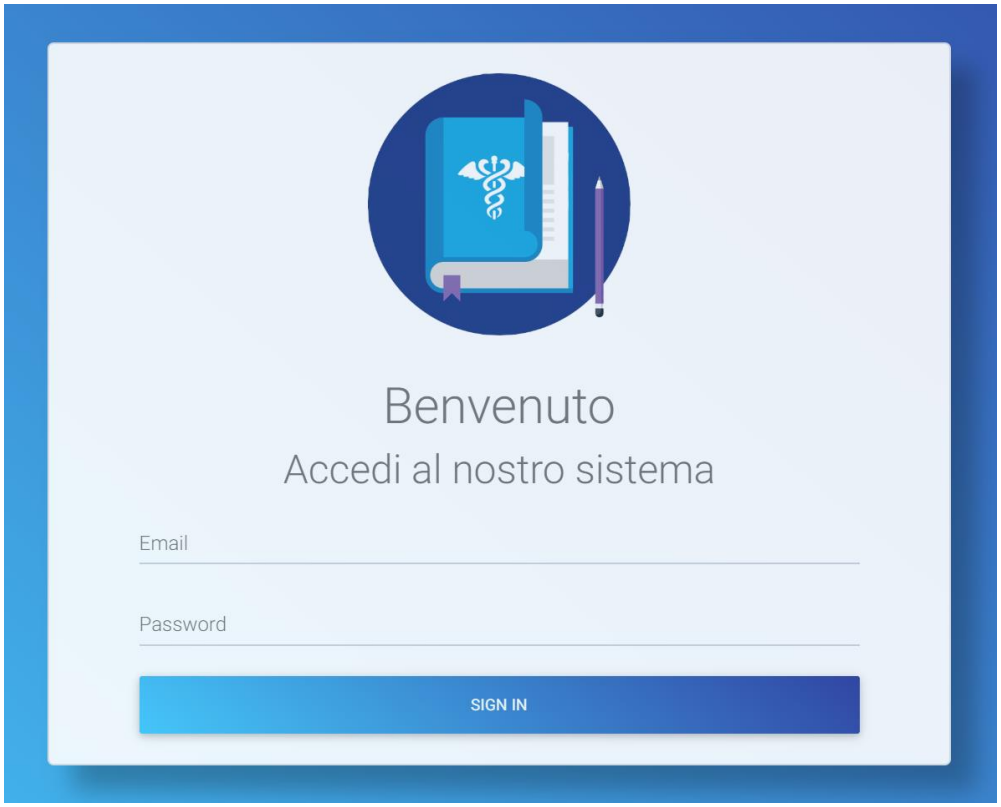


Figure 25 Pagina di login

Se le credenziali sono corrette si accede alla pagina di ricerca casi, altrimenti appare un messaggio di errore.

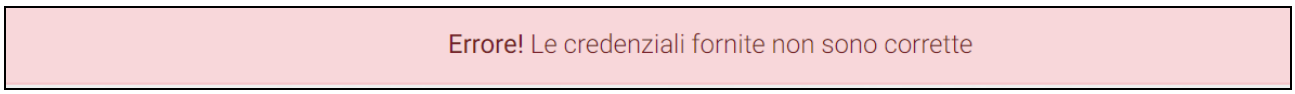


Figure 26 Messaggio credenziali sbagliate

3.4.2 Pagina di errore

Questa pagina viene mostrata quando l'utente non riesce a connettersi al database. È una semplice pagina che avverte l'utente che il tentativo di connessione al database è fallito.

In ogni metodo dell'applicazione (di interazione con l'utente) viene controllata la connessione al database e, se questa fallisce, viene chiamato il metodo statico **noDatabaseConnection()** del controller dbError.php.

Figure 27 Pagina errore di connessione alla banca dati

Nel metodo **noDatabaseConnection()** oltre a mostrare la pagina di errore viene richiamato il metodo statico **logout()** del controller users.php per evitare che un utente lasci la postazione tenendo la pagina aperta e qualcun altro, una volta che la connessione è stata stabilita, possa ricaricare la pagina ed effettuare operazioni con le credenziali del primo utente.

3.4.3 Pagina di ricerca casi

3.4.3.1 Filtro di ricerca

A questa pagina hano accesso tutti gli utenti che effettuano il login. Si possono visualizzare i vari casi e filtrarli in base ad alcune opzioni disponibili.

Filtri disponibili:

- Ordine:
  - Più recenti
  - Meno recenti
  - Più ricorrenti
- Testo
- Categoria
- Data specifica

Figure 28 Campi di ricerca

Tutte i filtri possono essere combinati tra di loro, si può quindi ad esempio effettuare una ricerca selezionando un tipo di ordinamento e una parola chiave.

Figure 29 Filtri combinati tra loro

Se una ricerca non produce nessun risultato viene stampato un messaggio che avvisa l'utente che nessun caso corrisponde ai criteri di ricerca che ha impostato.

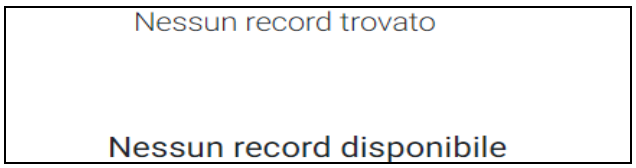


Figure 30 Nessun record trovato

I casi vengono mostrati con una semplice formattazione indicando le caratteristiche principali (nome, id, categoria, variante, data creazione, descrizione, numero di ripresentazioni). Ho preferito fare in modo che tutti i risultati vengano caricati una volta sola, per poi poter navigare velocemente e liberamente tra i casi. In questo modo impiega più tempo quando viene eseguita la query ma successivamente tramite il campo cerca tra i risultati si possono effettuare ricerche istantanee.

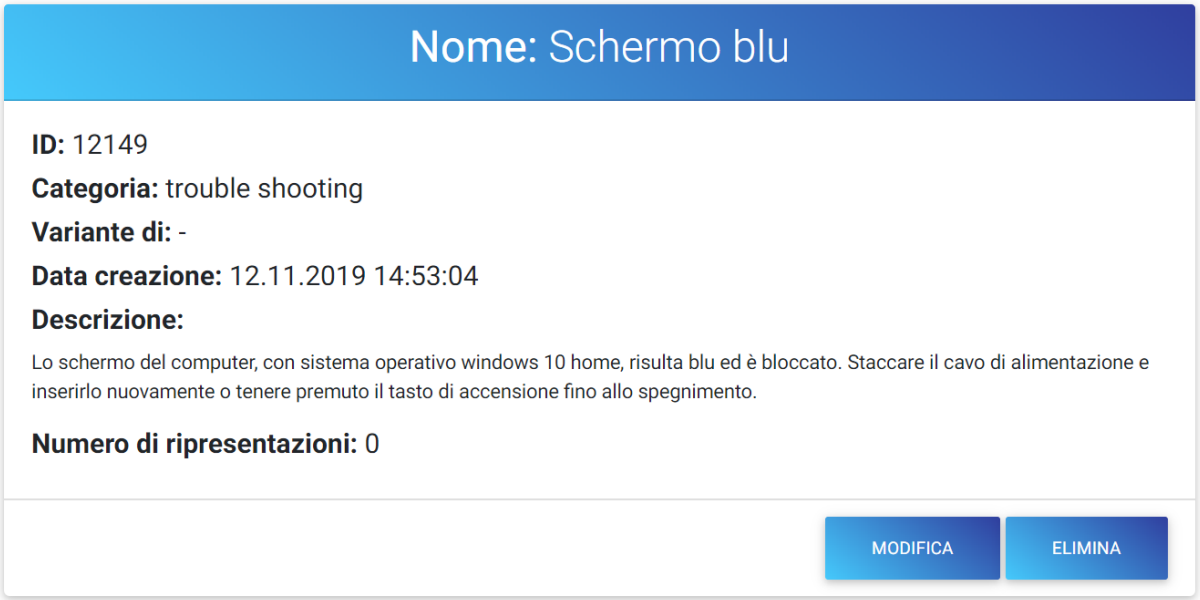


Figure 31 Esempio di caso di documentazione

3.4.3.2 Aggiunta e modifica caso

Qualunque utente può aggiungere un caso. Se si clicca sul bottone "**Aggiungi un caso"** si apre una finestra modale (che corrisponde a quella di modifica di un caso) in cui si deve inserire il titolo del caso, la categoria alla quale appartiene, se è una variante di un altro caso (selezionare il caso di cui è una variante) e la descrizione. Se la creazione dovesse fallire viene mostrato all'utente il relativo errore.

Se si clicca il bottone "modifica" su un caso (solo gli admin possono effettuare questa operazione) si apre una finestra modale in cui si possono modificare i dati del caso. Una volta inseriti i dati si può decidere se salvare le modifiche o meno.

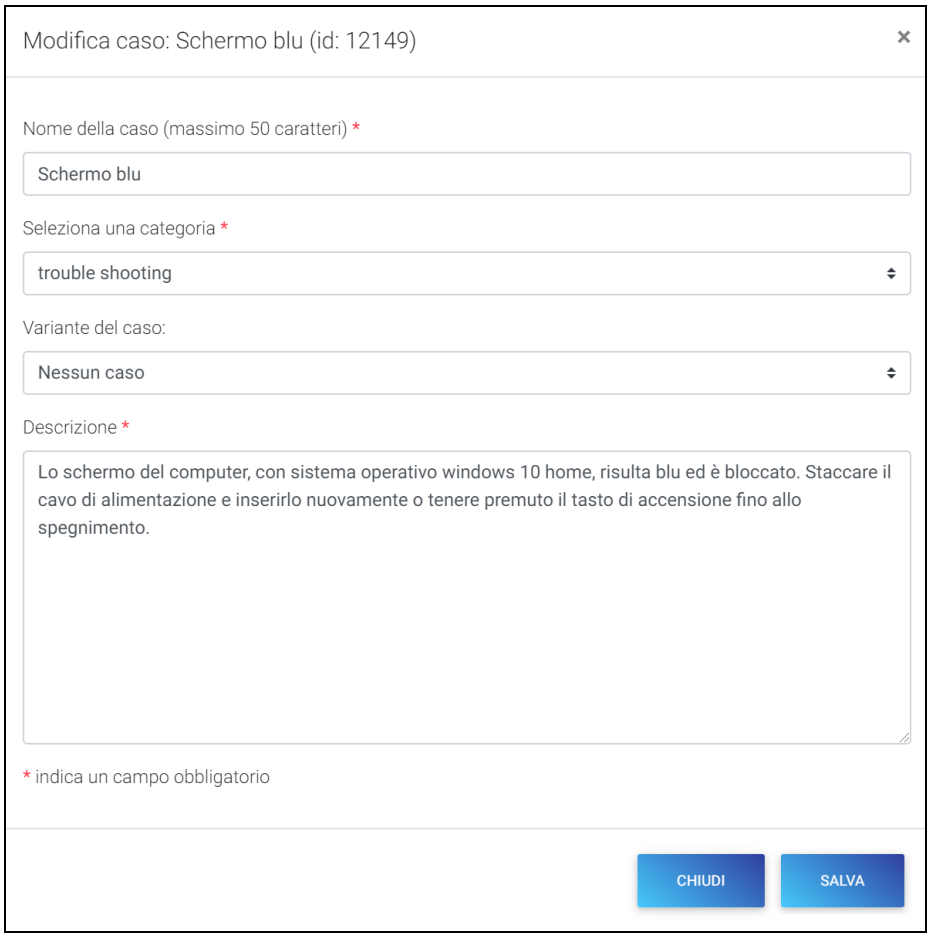


Figure 32 Finestra aggiunta/modifica caso

In caso di fallimento della modifica viene mostrato il messaggio con il relativo errore e i dati inseriti dall'utente rimangono in memoria. In questo modo l'utente non deve riscrivere i dati inseriti in precedenza.

3.4.3.3 Aggiunta, eliminazione categoria

Solo gli admin possono aggiungere o eliminare una categoria. Per aggiungere o eliminare una categoria basta cliccare sui relativi bottoni posti sotto la maschera dei filtri.

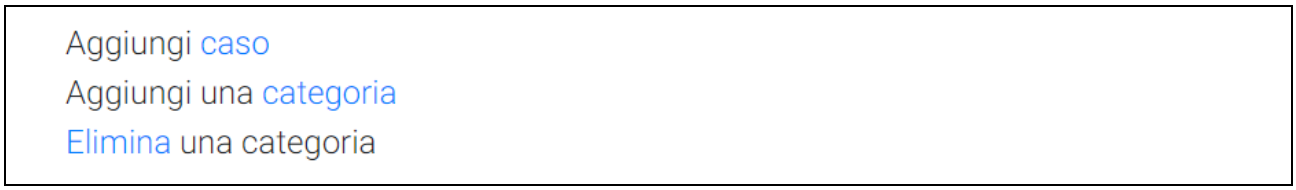


Figure 33 Bottoni funzionalità

Per aggiungere una categoria bisogna inserire un nome che non esista già (messaggio di errore se la categoria esiste già al momento della creazione) e che non superi i 50 caratteri.

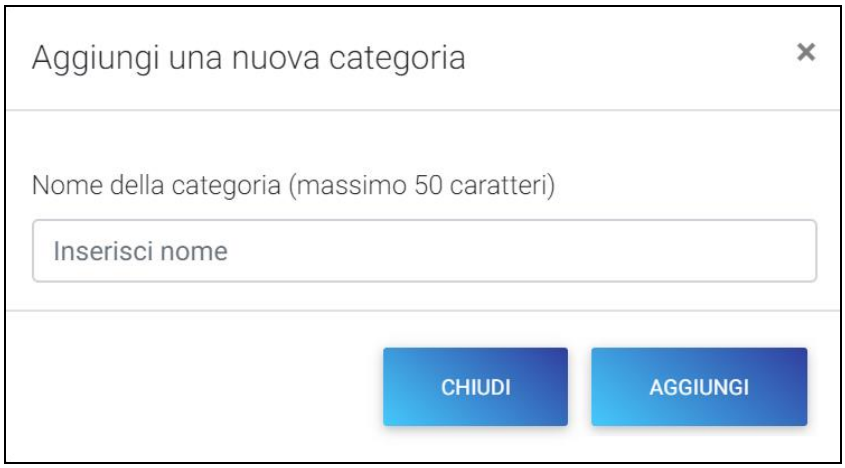


Figure 34 Finestra aggiunta categoria

Per eliminare una categoria basta selezionarla, premere su elimina e confermare l'eliminazione.

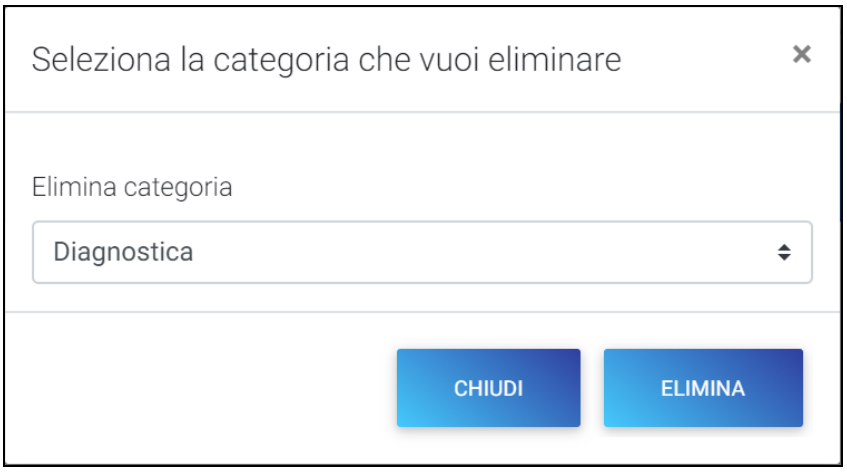


Figure 35 Finestra eliminazione categoria

3.4.4 Pagina gestione utenti

Solo l'admin può accedere a questa pagina in cui sono vengono mostrati gli utenti registrati. Si possono aggiungere nuovi utenti oppure rimuoverli/modificarli.

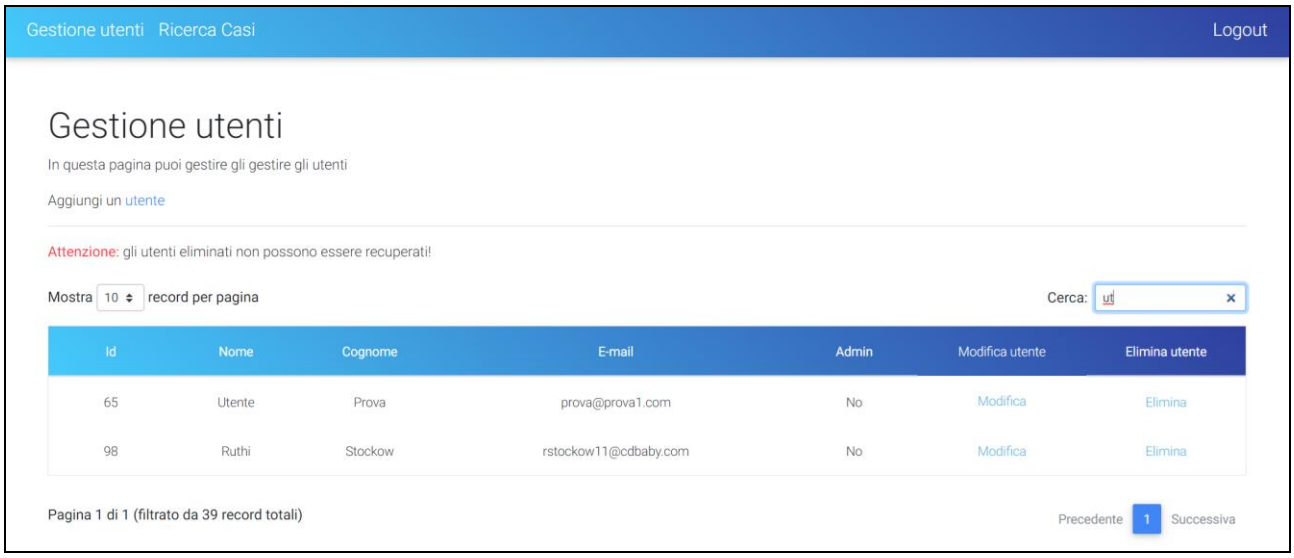


Figure 36 Pagina di gestione utenti

Quando si vuole aggiungere un nuovo utente si apre una finestra modale in cui inserire i dati dell'utente.

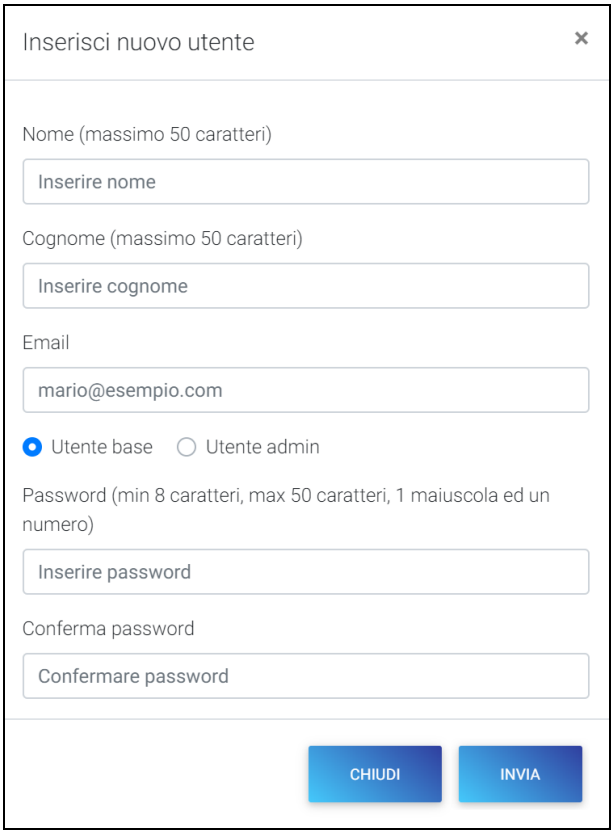


Figure 37 Finestra di aggiunta utente

Se durante la creazione avviene qualche errore viene stampato tramite un popup. I dati inseriti in precedenza non vanno persi, se si clicca di nuovo su crea un utente sono ancora presenti.

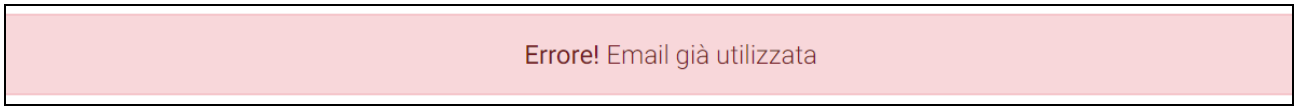


Figure 38 Messaggio di errore, email già utilizzata

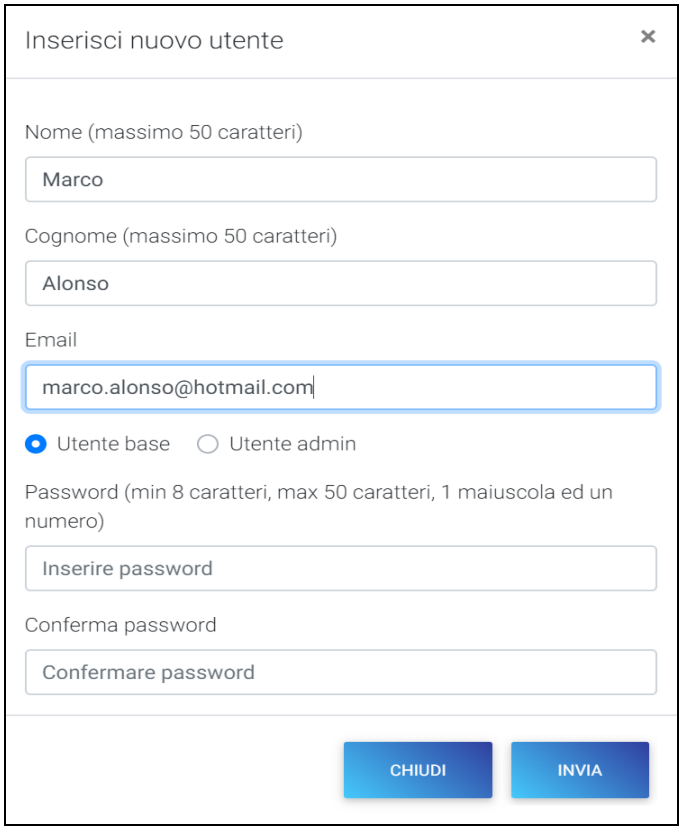


Figure 39 Finestra di aggiunta, informazioni conservate

Quando invece si desidera eliminare un utente basta premere sul **Elimina**. Prima di eliminare in modo definitivo un utente viene chiesta una conferma dell'azione tramite un modal.

Se un utente viene eliminato non vi è più nessun modo di recuperare lui ed i suoi dati.

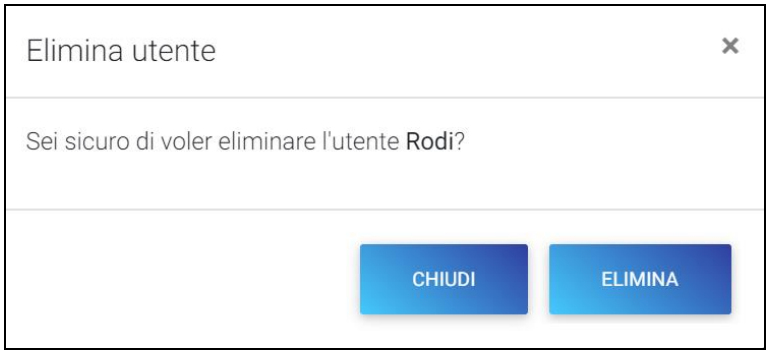


Figure 40 Conferma eliminazione utente

Premendo su “modifica” si apre una finestra modale di modifica dell’utente, in cui si può cambiare il nome, cognome, indirizzo email, privilegi e password (opzionale).

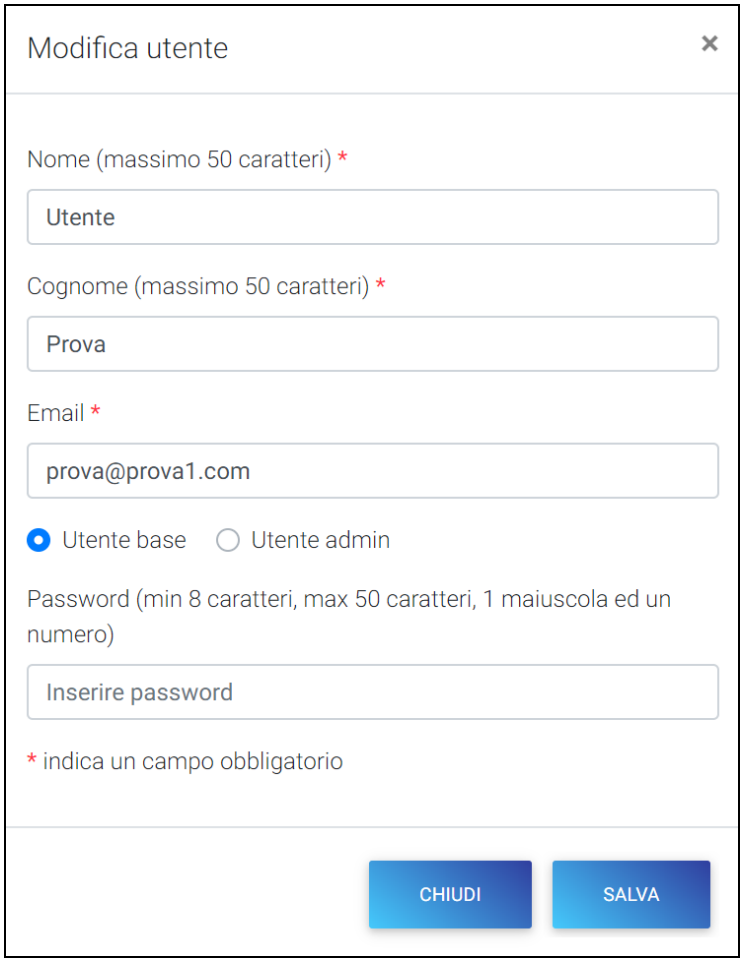


Figure 41 Finestra modifica utente

4 Test

4.1 Protocollo di test

Test Case: Riferimento:	TC-001 REQ-01 REQ-07	Nome:	Pagina di login
Descrizione:	Inserire credenziali di accesso non corrette		
Prerequisiti:	Pagina di login, sistema di login, database con presenti i dati		
Procedura:	1. Aprire il browser e cercare localhost:8080/knowledge_base 2. Inserire nei campi di testo nome utente e password non corretti 3. Premere il bottone “accedi”		
Risultati attesi:	Viene mostrato un messaggio di errore all’utente in cui viene comunicato che le credenziali inserite non sono corrette.		

Test Case: Riferimento:	TC-002 REQ-01 REQ-07	Nome:	Pagina di login
Descrizione:	Inserire credenziali di accesso corrette		
Prerequisiti:	Pagina di login, sistema di login, pagina di ricerca casi, database con presenti i dati		
Procedura:	1. Aprire il browser e cercare localhost:8080/knowledge_base 2. Inserire nei campi di testo nome utente e password corretti 3. Premere il bottone “accedi”		
Risultati attesi:	L’utente riesce ad accedere in modo corretto e reindirizzato nella pagina di ricerca casi.		

Test Case: Riferimento:	TC-003 REQ-02	Nome:	Visualizzazione utenti, pagina di gestione utenti (admin)
Descrizione:	Visualizzazione utenti registrati		
Prerequisiti:	Pagina di login, sistema di login, database con presenti i dati, pagina gestione utenti		
Procedura:	1. Aprire il browser e cercare localhost:8080/knowledge_base 2. Inserire nei campi di testo nome utente e password corretti di un amministratore 3. Premere il bottone “accedi” 4. Premere sul menu in alto la voce “Gestione utenti”		
Risultati attesi:	L’utente riesce a visualizzare gli utenti in modo corretto e può cercarli tramite il campo testuale cerca.		

Test Case: Riferimento:	TC-004 REQ-02 REQ-06	Nome:	Creazione nuovo utente, operazione fallita
Descrizione:	Tentativo fallito di creazione utente		
Prerequisiti:	Pagina di login, sistema di login, database con presenti i dati, pagina gestione utenti		
Procedura:	1. Aprire il browser e cercare localhost:8080/knowledge_base 2. Inserire nei campi di testo nome utente e password corretti di un amministratore 3. Premere il bottone “accedi” 4. Premere sul menu in alto la voce “Gestione utenti” 5. Premere su “Aggiungi un utente” 6. Inserire i dati rispettando la sintassi 7. Inserire due password diverse 8. Premere “crea” 9. Ripetere l’operazione non rispettando la sintassi per ogni campo richiesto (vedi specifiche per ogni campo richiesto nel testo sopra all’input)		
Risultati attesi:	All’utente appare un messaggio di errore esplicativo.		

Test Case:	TC-005	Nome:	Creazione nuovo utente, operazione riuscita
Riferimento:	REQ-02 REQ-06		
Descrizione:	Tentativo fallito di creazione utente		
Prerequisiti:	Pagina di login, sistema di login, database con presenti i dati, pagina gestione utenti		
Procedura:	1. Aprire il browser e cercare localhost:8080/knowledge_base 2. Inserire nei campi di testo nome utente e password corretti di un amministratore 3. Premere il bottone “accedi” 4. Premere sul menu in alto la voce “Gestione utenti” 5. Premere su “Aggiungi un utente” 6. Inserire i dati rispettando la sintassi dei vari campi 7. Premere “crea”		
Risultati attesi:	All’utente appare un messaggio in cui viene informato che l’operazione ha avuto successo. L’utente appare nella lista degli utenti.		

Test Case:	TC-006	Nome:	Eliminazione utente, operazione riuscita
Riferimento:	REQ-02		
Descrizione:	Eliminazione di un utente		
Prerequisiti:	Pagina di login, sistema di login, pagina di ricerca casi, database con presenti i dati, pagina gestione utenti		
Procedura:	1. Aprire il browser e cercare localhost:8080/knowledge_base 2. Inserire nei campi di testo nome utente e password corretti di un amministratore 3. Premere il bottone “accedi” 4. Premere sul menu in alto la voce “Gestione utenti” 5. Premere su “Elimina” sull’utente che si desidera eliminare 6. Confermare l’eliminazione		
Risultati attesi:	All’utente appare un messaggio in cui viene informato che l’operazione ha avuto successo. L’utente non appare nella lista degli utenti.		

Test Case: Riferimento:	TC-007 REQ-02	Nome:	Eliminazione utente, operazione fallita
Descrizione:	Eliminazione del l'utente con cui si è loggati		
Prerequisiti:	Pagina di login, sistema di login, pagina di ricerca casi, database con presenti i dati, pagina gestione utenti		
Procedura:	1. Aprire il browser e cercare localhost:8080/knowledge_base 2. Inserire nei campi di testo nome utente e password corretti di un amministratore 3. Premere il bottone "accedi" 4. Premere sul menu in alto la voce "Gestione utenti" 5. Premere "Elimina" sull'utente con cui si è loggati 6. Confermare l'eliminazione		
Risultati attesi:	All'utente appare un messaggio di errore in cui viene avvisato che non può eliminare il proprio account. L'operazione non è andata a buon fine.		

Test Case: Riferimento:	TC-008 REQ-03	Nome:	Visualizzazione casi
Descrizione:	Visualizzazione casi presenti nel sistema		
Prerequisiti:	Pagina di login, sistema di login, pagina di ricerca casi, database con presenti i dati		
Procedura:	1. Aprire il browser e cercare localhost:8080/knowledge_base 2. Inserire nei campi di testo nome utente e password corretti 3. Premere il bottone "accedi" 4. Aspettare che la pagina sia caricata		
Risultati attesi:	In questa pagina vengono mostrati all'utente i casi presenti nel sistema. Tramite il campo di testo "cerca in live" l'utente può cercare in modo dinamico i casi inserendo del testo.		

Test Case:	TC-009	Nome:	Filtraggio casi
Riferimento:	REQ-03 REQ-05		
Descrizione:	Filtraggio casi		
Prerequisiti:	Pagina di login, sistema di login, pagina di ricerca casi, database con presenti i dati		
Procedura:	1. Aprire il browser e cercare localhost:8080/knowledge_base 2. Inserire nei campi di testo nome utente e password corretti 3. Premere il bottone “accedi” 4. Aspettare che la pagina sia caricata 5. Premere sul bottone “Mostra filtri” 6. Inserire il tipo di ordinamento desiderato 7. Inserire il testo da cercare 8. Inserire la categoria del caso che si vuol filtrare 9. Inserire la data di creazione del caso 10. Premere il bottone “cerca”		
Risultati attesi:	All’utente vengono mostrati i casi che corrispondono ai filtri applicati in precedenza. Tramite il campo di testo “cerca in live” l’utente può cercare in modo dinamico tra i casi filtrati inserendo del testo.		

Test Case:	TC-010	Nome:	Aggiunta caso, operazione fallita
Riferimento:	REQ-03		
Descrizione:	Aggiunta di un caso, non inserendo i dati rispettando la sintassi		
Prerequisiti:	Pagina di login, sistema di login, pagina di ricerca casi, database con presenti i dati		
Procedura:	1. Aprire il browser e cercare localhost:8080/knowledge_base 2. Inserire nei campi di testo nome utente e password corretti 3. Premere il bottone “accedi” 4. Aspettare che la pagina sia caricata 5. Premere su “Aggiungi caso” 6. Inserire nei campi “nome” oppure “descrizione” il carattere “spazio” 7. Inserire gli altri dati rispettando la sintassi 8. Premere “Crea”		
Risultati attesi:	All’utente viene mostrato un messaggio di errore che indica che i campi testuali devono contenere del testo.		

Test Case:	TC-011	Nome:	Aggiunta caso, operazione riuscita
Riferimento:	REQ-03		
Descrizione:	Aggiunta di un caso, inserendo i dati rispettando la sintassi		
Prerequisiti:	Pagina di login, sistema di login, pagina di ricerca casi, database con presenti i dati		
Procedura:	1. Aprire il browser e cercare localhost:8080/knowledge_base 2. Inserire nei campi di testo nome utente e password corretti 3. Premere il bottone “accedi” 4. Aspettare che la pagina sia caricata 5. Premere su “Aggiungi caso” 6. Riempire i dati con dei valori che rispettano la sintassi 7. Premere “Crea”		
Risultati attesi:	All’utente viene mostrato un messaggio di successo che indica che il caso è stato creato con successo. Il caso appare nella lista dei casi.		

Test Case:	TC-012	Nome:	Eliminazione caso
Riferimento:	REQ-03		
Descrizione:	Eliminazione di un caso		
Prerequisiti:	Pagina di login, sistema di login, pagina di ricerca casi, database con presenti i dati		
Procedura:	1. Aprire il browser e cercare localhost:8080/knowledge_base 2. Inserire nei campi di testo nome utente e password corretti di un amministratore 3. Premere il bottone “accedi” 4. Premere sul bottone “elimina” sul caso desiderato 5. Premere “Chiudi” per annullare l’eliminazione o “Elimna” per confermare l’eliminazione		
Risultati attesi:	Se l’utente non conferma l’eliminazione e preme “Chiudi” il caso non viene eliminato. Altrimenti, se l’utente preme “Elimina” il caso viene eliminato, viene mostrato un messaggio di successo ed il caso viene eliminato. Il caso non appare più tra la lista di casi.		

Test Case: Riferimento:	TC-013 REQ-03 REQ-08	Nome:	Aggiunta di una categoria, operazione fallita
Descrizione:	Aggiunta di una categoria, operazione fallita		
Prerequisiti:	Pagina di login, sistema di login, pagina di ricerca casi, database con presenti i dati		
Procedura:	1. Aprire il browser e cercare localhost:8080/knowledge_base 2. Inserire nei campi di testo nome utente e password corretti di un amministratore 3. Premere il bottone “Aggiungi categoria” 4. Inserire il nome di una categoria già esistente 5. Premere il bottone “crea”		
Risultati attesi:	All’utente viene mostrato un messaggio di errore che avvisa l’utente che la categoria esiste già. L’operazione fallisce.		

Test Case: Riferimento:	TC-014 REQ-03 REQ-08	Nome:	Aggiunta di una categoria, operazione riuscita
Descrizione:	Aggiunta di una categoria, operazione riuscita		
Prerequisiti:	Pagina di login, sistema di login, pagina di ricerca casi, database con presenti i dati		
Procedura:	1. Aprire il browser e cercare localhost:8080/knowledge_base 2. Inserire nei campi di testo nome utente e password corretti di un amministratore 3. Premere il bottone “Aggiungi categoria” 4. Inserire il nome di una categoria che non esistente già 5. Premere il bottone “crea”		
Risultati attesi:	All’utente viene mostrato un messaggio di successo che avvisa l’utente che la categoria è stata creata con successo. L’operazione è riuscita.		

Test Case: Riferimento:	TC-015 REQ-03	Nome:	Eliminazione categoria
Descrizione:	Eliminazione categoria		
Prerequisiti:	Pagina di login, sistema di login, pagina di ricerca casi, database con presenti i dati		
Procedura:	1. Aprire il browser e cercare localhost:8080/knowledge_base 2. Inserire nei campi di testo nome utente e password corretti di un amministratore 3. Premere il bottone “Eliminazione categoria” 4. Selezionare la categoria che si desidera eliminare 5. Premere il bottone “Elimina” 6. Premere il bottone “Chiudi” per annullare l’operazione o “Elimina” per eliminare la categoria		
Risultati attesi:	Se l’utente preme “Chiudi” l’azione viene annullata. Se l’utente preme “Elimina” la categoria viene eliminata e viene mostrato un messaggio di operazione riuscita all’utente.		

4.2 Risultati test

Codice-Test	Risultato	Note
TC-01	Passato	
TC-02	Passato	
TC-03	Passato	
TC-04	Passato	
TC-05	Passato	
TC-06	Passato	
TC-07	Passato	
TC-08	Passato	
TC-09	Passato	
TC-10	Passato	
TC-11	Passato	
TC-12	Passato	
TC-13	Passato	
TC-14	Passato	
TC-15	Passato	

4.3 Mancanze/limitazioni conosciute

L'applicativo presenta alcune limitazioni. Il layout dei "casi" potrebbe essere migliorato, in modo da facilitare la lettura e la navigazione. Potrebbe essere aggiunto un bottone che rimanda al caso padre quando viene premuto.

Manca una visualizzazione riassuntiva dei casi, un piccolo elenco in cui vengono mostrati i titoli dei casi e premendoci sopra si viene riportati al caso desiderato.

Nell'applicazione manca inoltre un sistema di recupero autonomo della password. Al momento solo gli amministratori possono modificare le informazioni personali degli utenti, ovvero nome, cognome, email, tipo di utente e password.

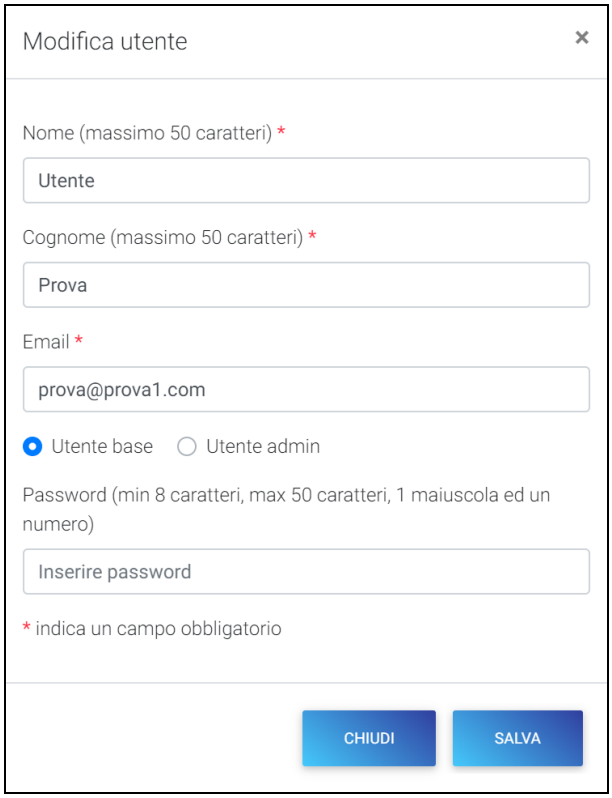


Figure 42 Finestra modifica utente

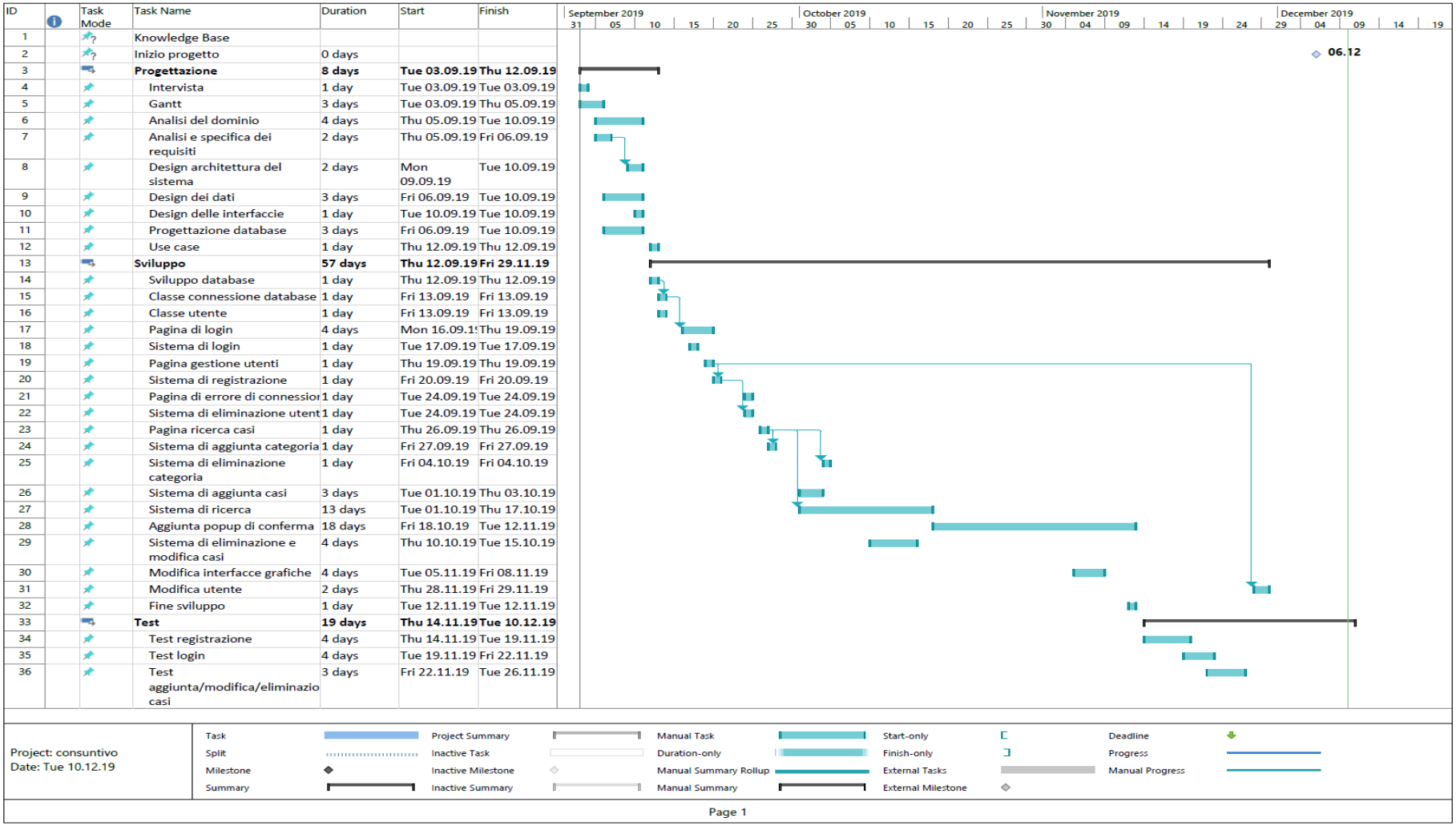
5 Consuntivo

Il consuntivo finale non è molto diverso da quello che ho pianificato nella fase di analisi, ci sono state attività che sono durate leggermente più del previsto ed altre di meno.

Alcune attività risultano più lunghe perché sono state effettuate delle modifiche in un secondo momento oppure sono state lasciate delle parti in sospeso. Sono state inoltre aggiunte delle task che nel preventivo le avevo considerate come una parte di un'altra attività, quindi nel consuntivo risultano un maggior numero di attività rispetto al preventivo per questo motivo.

Durante le lezioni ho seguito la mia pianificazione passo per passo e mi è stato molto utile in quanto i tempi corrispondevano quasi durante tutto il progetto. Sono riuscito ad implementare tutte le funzionalità che mi sono state richieste dal committente.

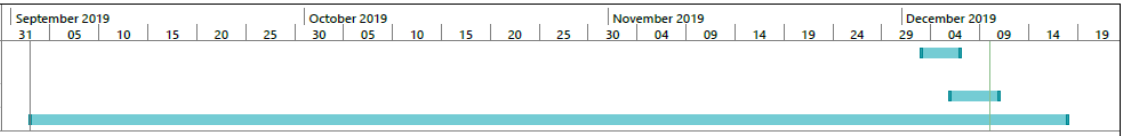
In conclusione il consuntivo corrisponde quasi al preventivo se non con alcune variazioni di durate delle attività.



ID	Task Mode	Task Name	Duration	Start	Finish
1		Knowledge Base			
2		Inizio progetto	0 days		
3		**Progettazione**	**8 days**	**Tue 03.09.19**	**Thu 12.09.19**
4		Intervista	1 day	Tue 03.09.19	Tue 03.09.19
5		Gantt	3 days	Tue 03.09.19	Thu 05.09.19
6		Analisi del dominio	4 days	Thu 05.09.19	Tue 10.09.19
7		Analisi e specifica dei requisiti	2 days	Thu 05.09.19	Fri 06.09.19
8		Design architettura del sistema	2 days	Mon 09.09.19	Tue 10.09.19
9		Design dei dati	3 days	Fri 06.09.19	Tue 10.09.19
10		Design delle interfaccie	1 day	Tue 10.09.19	Tue 10.09.19
11		Progettazione database	3 days	Fri 06.09.19	Tue 10.09.19
12		Use case	1 day	Thu 12.09.19	Thu 12.09.19
13		**Sviluppo**	**57 days**	**Thu 12.09.19**	**Fri 29.11.19**
14		Sviluppo database	1 day	Thu 12.09.19	Thu 12.09.19
15		Classe connessione database	1 day	Fri 13.09.19	Fri 13.09.19
16		Classe utente	1 day	Fri 13.09.19	Fri 13.09.19
17		Pagina di login	4 days	Mon 16.09.1	Thu 19.09.19
18		Sistema di login	1 day	Tue 17.09.19	Tue 17.09.19
19		Pagina gestione utenti	1 day	Thu 19.09.19	Thu 19.09.19
20		Sistema di registrazione	1 day	Fri 20.09.19	Fri 20.09.19
21		Pagina di errore di connessior	1 day	Tue 24.09.19	Tue 24.09.19
22		Sistema di eliminazione utent	1 day	Tue 24.09.19	Tue 24.09.19
23		Pagina ricerca casi	1 day	Thu 26.09.19	Thu 26.09.19
24		Sistema di aggiunta categoria	1 day	Fri 27.09.19	Fri 27.09.19
25		Sistema di eliminazione categoria	1 day	Fri 04.10.19	Fri 04.10.19
26		Sistema di aggiunta casi	3 days	Tue 01.10.19	Thu 03.10.19
27		Sistema di ricerca	13 days	Tue 01.10.19	Thu 17.10.19
28		Aggiunta popup di conferma	18 days	Fri 18.10.19	Tue 12.11.19
29		Sistema di eliminazione e modifica casi	4 days	Thu 10.10.19	Tue 15.10.19
30		Modifica interfacce grafiche	4 days	Tue 05.11.19	Fri 08.11.19
31		Modifica utente	2 days	Thu 28.11.19	Fri 29.11.19
32		Fine sviluppo	1 day	Tue 12.11.19	Tue 12.11.19
33		**Test**	**19 days**	**Thu 14.11.19**	**Tue 10.12.19**
34		Test registrazione	4 days	Thu 14.11.19	Tue 19.11.19
35		Test login	4 days	Tue 19.11.19	Fri 22.11.19
36		Test aggiunta/modifica/eliminazio casi	3 days	Fri 22.11.19	Tue 26.11.19

Titolo del progetto: Knowledge Base
Alunno/a: Bryan Beffa
Classe: Info I4AA
Anno scolastico: 2019/2020
Docente responsabile: Ivan Raimondi

ID	Task Mode	Task Name	Duration	Start	Finish
37		Test aggiunta/eliminazione categoria	4 days	Tue 03.12.19	Fri 06.12.19
38		Test ricerca	3 days	Fri 06.12.19	Tue 10.12.19
39		Documentazione	76 days	Tue 03.09.19	Tue 17.12.19



Project: consuntivo
Date: Tue 10.12.19

Task
Split
Milestone
Summary
Project Summary
Inactive Task
Inactive Milestone
Inactive Summary
Manual Task
Duration-only
Manual Summary Rollup
Manual Summary
Start-only
Finish-only
External Tasks
External Milestone
Deadline
Progress
Manual Progress

Page 2

6 Conclusioni

6.1 Sviluppi futuri

Ho pensato a molti sviluppi futuri per questa applicazione. Come prima cosa inserirei un sistema di recupero password autonomo con l'utilizzo di un server mail. In questo modo quando un utente ha bisogno di effettuare un cambio password non deve recarsi dagli amministratori ma lo può effettuare individualmente ricevendo un codice per cambiare la password per email.

Si potrebbe aggiungere il cambio di informazioni personali autonomo, anche in questo caso non si dovrebbe più andare dall'amministratore per cambiare ad esempio indirizzo email.

Come consigliato dal docente Raimondi aggiungerei un layout riassuntivo dei casi di documentazione, così da potersi muovere cliccando sul caso desiderato direttamente dalla finestra riassuntiva.

6.2 Considerazioni personali

Sono soddisfatto del prodotto finale. Questa applicazione web svolge le funzioni richieste dal committente e può essere facilmente portata avanti da altre persone. Ho cercato di rendere l'applicazione user-friendly elaborando layout semplici e intuitivi.

Personalmente ho trovato il progetto interessante e mi ha permesso di testare ciò che ho appreso durante la mia formazione professionale. Se avessi avuto più tempo a disposizione avrei potuto aggiungere più funzionalità all'applicazione web.

7 Bibliografia

Per la realizzazione di questo progetto non ho consultato riviste, libri, manuali ma solo i siti web descritti nel capitolo sitografia.

7.1 Sitografia

https://www.draw.io/, Draw.io, dal 05.09.2018 al

https://mdbootstrap.com/docs/jquery/getting-started/download/, material design bootstrap download, 17.09.2019

https://mdbootstrap.com/docs/jquery/forms/basic/, form base di material design, 17.09.2019

https://mdbootstrap.com/docs/jquery/navigation/navbar/, navbar material design, 19.09.2019

https://mdbootstrap.com/docs/jquery/components/alerts/, material design alert, 20.09.2019

https://stackoverflow.com/questions/, stackoverflow, 15.10.2019 – 20.12.2019

https://mockaroo.com/, mackaroo creazione dati, 25.10.2019

8 Allegati

Elenco degli allegati:

- Quaderno dei compiti
- Guida installazione XDebug
- Diari di lavoro
- Codici sorgente, git https://github.com/bryanbeffa/Knowledge-Base