### Sommario

[Sommario 1](#_Toc477429968)

[INTRODUZIONE 1](#_Toc477429969)

[CMS 2](#_Toc477429970)

[WCMS 2](#_Toc477429971)

[DCMS 2](#_Toc477429972)

[Mik CMS 3](#_Toc477429973)

[Descrizione del Problema 4](#_Toc477429974)

[Architettura Software 4](#_Toc477429975)

[Tecnologie utilizzate 5](#_Toc477429976)

[Balsamiq 5](#_Toc477429977)

[HTML 5 5](#_Toc477429978)

[CSS 3 6](#_Toc477429979)

[Bootstrap 3 6](#_Toc477429980)

[Jquery 6](#_Toc477429981)

[MySQL 7](#_Toc477429982)

[Apache 7](#_Toc477429983)

[PHP 7](#_Toc477429984)

[Organizzazione Base Dati 7](#_Toc477429985)

[categories 8](#_Toc477429986)

[css\_wys 8](#_Toc477429987)

[docs\_it 9](#_Toc477429988)

[languages 9](#_Toc477429989)

[media 10](#_Toc477429990)

[users 10](#_Toc477429991)

[Organizzazione del File System 11](#_Toc477429992)

[Panoramica del progetto 11](#_Toc477429993)

[Indice documentazione (Homepage) 12](#_Toc477429994)

[Menù laterale 15](#_Toc477429995)

[Categorie API 19](#_Toc477429996)

[Media 23](#_Toc477429997)

[Documenti API 28](#_Toc477429998)

[pagine.php 28](#_Toc477429999)

[articolo.php 29](#_Toc477430000)

[tinyMCE: l’editor WYSIWYG 33](#_Toc477430001)

[pulsanti personalizzati 33](#_Toc477430002)

[plugin “import css” 34](#_Toc477430003)

[CSS WYSIWYG 36](#_Toc477430004)

[WCMS : Sezione Live 44](#_Toc477430005)

[Menù 44](#_Toc477430006)

[Pagine Menù 45](#_Toc477430007)

[Struttura 46](#_Toc477430008)

[Sviluppi futuri: supporto multilingua 47](#_Toc477430009)

# 

# INTRODUZIONE

L’obbiettivo di questa tesi è documentare la progettazione e la realizzazione di un **DCMS** ( Dynamic Content Management System) che permetta la produzione di documentazione digitale in maniera semplice e guidata. Il **DCMS** in questione darà quindi la possibilità di creare della documentazione che sarà memorizzata in una banca dati e potrà essere visualizzata dall’utente tramite il portale riservato a quest’ultimo.

L’intero progetto è suddiviso in 3 moduli:

* il primo è modulo è il **DCMS**, utilizzato dagli sviluppatori per produrre la documentazione
* il secondo modulo è il **WCMS**, utilizzato dagli sviluppatori per la gestione web del portale dedicato all’utenza
* il terzo modulo è il portale che permette agli utenti di usufruire della documentazione

Il **DCMS** è stato progettato per la produzione della documentazione del software **Mik\_cms**.

## CMS

In informatica un Content Management System, in acronimo **CMS** (sistema di gestione dei contenuti in italiano), è uno strumento che risiede su un server web e permette la gestione dei contenuti in maniera guidata, semplice e intuitiva.

In questo modo svincoliamo il webmaster, o chi per lui gestisce i contenuti, da conoscenze tecniche, potendo quest’ultimo utilizzare delle procedure guidate per manipolare i contenuti.

## WCMS

L’utilizzo più frequente dei **CMS** è sicuramente quello della gestione di siti web. L’avvento dei **CMS** ha permesso la produzione di siti web, anche di grandi dimensioni, in tempi ristretti. Non è solo il processo di produzione ad averne tratto beneficio, ma soprattutto quello di manutenzione dei siti web. Infatti tramite un sistema di *Web Content Management System* (**WCMS**), possiamo non solo modificare l’interfaccia e quindi anche i pilastri del sistema stesso, ma anche gestire i contenuti di cui l’utente va ad usufruire; e tutto questo avviene in maniera intuitiva e guidata.

I **WCMS** si rivelano l’arma vincente proprio quando i siti web diventano portali, ovvero quando la loro complessità e la loro dimensione cominciano a crescere. Ad esempio un portale per l’e-commerce dovrà gestire molti aspetti che in un normale sito vetrina non troveremmo, come ad esempio: gestione della registrazioni di utenti, gestione del carrello, gestione pagamenti elettronici e molto altro.

Alla luce di tutte queste esigenze, si rende necessario l’utilizzo di uno strumento che renda il lavoro di gestione del portale e dei suoi contenuti più semplice.

Gli esempi più famosi di WCMS sono Joomla, WordPress e Drupal.

## DCMS

Il *Dynamic Content Management System* (**DCMS**), oltre alle funzioni tipiche del **CMS**, offre la possibilità di generare della documentazione in maniera dinamica. Quindi oltre alla gestione dei contenuti che troviamo anche nel **WCMS**, come immagini, testo e tabelle, nel **DCMS** è possibile la gestione e la produzione di documentazione.

Il **DCMS** trova applicazione soprattutto nell'ambito industriale, come strumento interno di produzione e gestione della documentazione.

# Mik CMS

In questo capitolo affronteremo passo dopo passo le fasi di progettazione e produzione che hanno interessato la creazione del CMS.

## Descrizione del Problema

Per permettere ad aziende terze di utilizzare il software **Mik\_cms** nel pieno delle sue potenzialità, è sorta la necessità di produrre una documentazione adeguata; trattandosi di un software che offre una vasta quantità di funzionalità, da subito risulta non banale interfacciarsi con quest’ultimo.

Essendo un prodotto digitale destinato alla distribuzione su scala internazionale, vi è la necessità di fornire documentazione accessibile a tutti i potenziali clienti, la quale dovrà inoltre essere quindi fornita in più lingue. Si è scelto, come è solito in questi casi, di mettere a disposizione la documentazione tramite un portale web, il quale oltre la documentazione potrà anche fornire informazioni addizionali all’utente.

Come già anticipato, **Mik\_cms** offre una vasta gamma di funzionalità, oltre la possibilità di essere integrato ed interfacciarsi con software di terze parti, quindi la documentazione dovrà risultare chiara ed esaustiva documentando ogni sua funzionalità in modo da offrire al cliente un solido manuale con cui potersi interfacciare riducendo al minimo la necessità di un’assistenza diretta da parte dell’azienda produttrice del software.

Nasce quindi la necessità di avere uno strumento che faciliti la produzione della documentazione e la gestione del portale che permetterà all’utente di usufruirne.

Sono proprio questi gli obbiettivi di **Mik\_cms** :

1. offrire uno strumento che renda la stesura della documentazione agevole .
2. permettere la gestione web del portarle che utilizzerà il potenziale utente.

## Architettura Software

**Mik\_cms** è composto da 3 moduli, di cui 2 dedicati agli amministratori ed uno all’utenza.   
Precisamente:

1. **Modulo DCMS:** dedicato agli amministratori. Produzione della documentazione.
2. **Modulo WCMS:** dedicato agli amministratori. Gestione Web del portale.
3. **Modulo View:** dedicato agli utenti. Portale web accessibile a tutti dove è possibile visualizzare la documentazione ed altri dati.

Di seguito parleremo delle funzionalità dei singoli moduli e di come queste siano state implementate, ma prima elencheremo tutte le tecnologie utilizzare per la produzione di **Mik\_cms**.

## Tecnologie utilizzate

Ho prodotto personalmente l’intero software **Mik\_cms**, occupandomi sia della parte *backend* sia di quella *frontend*, passando per la progettazione grafica. Non ho utilizzato nessun framework *PHP*, ma ho optato per la progettazione e produzione del software partendo da zero.

Questa scelta è dovuta a vari motivi:

1. **Didattico**. Il primo e più rilevante motivo è quello per cui partendo da zero avrei avuto l’occasione di apprendere le basi di progettazione di un software simile. Ho consolidato le mie conoscenze nell’ambito di progettazione di basi dati, il modo in cui organizzare il file system per permettere di avere modello che ricordasse l’*MVC*. Infatti il software non può essere definito un *MVC* vero e proprio in quanto ho scelto un approccio alla programmazione procedurale. Ho comunque rispettato i principi base del modello *MVC*, come ad esempio la separazione della logica dalla parte di presentazione.
2. **Personalizzazione**. Se avessi utilizzato uno dei tanti **CMS** in circolazione (ad esempio *Wordpress, Drupal* o *Joomla*) avrei avuto una miriade di funzionalità non utili agli obbiettivi finali del software, non avendo contemporaneamente funzionalità utili alla produzione di documentazione. In definitiva partire da zero, senza alcun vincolo, mi ha permesso di progettare e produrre un software cucito sulle esigenze dell’azienda, che adempie perfettamente ai doveri prefissati.
3. **Sicurezza**. Se avessi utilizzato uno dei **CMS** disponibili o uno dei qualsiasi framework *PHP* disponibili (ad esempio *Sympony*) avrei esposto il progetto a delle eventuali vulnerabilità già note degli stessi. Invece non essendo noto il file system ed avendo adottato accorgimenti per la sicurezza, il software è stato reso sicuro.

### Balsamiq

“Balsamiq Mockups is a rapid wireframing tool that helps you Work Faster & Smarter. It reproduces the experience of sketching on a whiteboard, but using a computer.” - balsamiq.com

*Balsamiq* è un tool che permette una creazione rapida ed intuitiva di mockups. È stato utilizzato per la progettazione grafica, che è stato il primo step della fase di progettazione, permettendo di stabilire chiaramente a priori quali sono le idee di progettazione senza rischio di incomprensioni.

### HTML 5

L'*HTML* è un linguaggio di markup per la strutturazione delle pagine web.

La versione 5, ovvero l’*HTML5*, utilizzata in questo progetto, offre delle novità rispetto all’*HTML4* che sono finalizzate a migliorare la separazione fra la struttura definita dal markup (es. *<body>*), le caratteristiche di resta (colore, famiglia della font) definite probabilmente da un foglio *CSS* ed i reali contenuti della pagina web, come ad esempio il testo vero e proprio.

Un’altra importante novità introdotta dall’*HTML5* è comunque la possibilità di memorizzare localmente una grossa quantità di dati scaricati dal browser in modo da consentire l’utilizzo di applicazioni basate sul web and in assenza di connettività, come ad esempio *Gmail*.

Quindi l*’HTML5* può essere definito come il pilastro della parte frontend, utilizzato per definire la struttura di ogni pagina. Trattandosi della versione 5 sono stati utilizzati tag specifici di questa versione come *<code>* e *<section>*.

### CSS 3

Il *CSS* è stato utilizzato per abbellire l’intera struttura creata in *HTML*.

Il *CSS* (*Cascading Style Sheets*), è un linguaggio utilizzato per definire la formattazione di documenti *HTML*.

L'introduzione del *CSS* si è resa necessaria per separare i contenuti delle pagine HTML dalla loro formattazione e permettere una programmazione più chiara e facile da utilizzare, sia per gli autori delle pagine stesse sia per gli utenti, garantendo contemporaneamente anche il riutilizzo di codice ed una sua più facile manutenzione.

### Bootstrap 3

“Bootstrap is the most popular HTML, CSS, and JS framework for developing responsive, mobile first projects on the web.

Bootstrap makes front-end web development faster and easier. It's made for folks of all skill levels, devices of all shapes, and projects of all sizes.” - getbootstrap.com

Per la realizzazione del progetto, oltre al semplice *HTML*, si è scelto di utilizzare il framework *Bootstrap*. Utilizzando questo framework, si è reso il progetto responsive e quindi fruibile anche da dispositivi mobile. *Bootstrap* si è rivelato un componente chiave della produzione, in quanto offrendo tantissimi widgets, ha reso il tutto più semplice, veloce e soprattutto armonioso graficamente.

In particolare è stato utilizzato il template “flatty”.

### Jquery

L’utilizzo di *Javascript* è stato determinante per rendere la piattaforma dinamica e pronta ad adattarsi in base alle scelte dell’utente. L’utilizzo di *Jquery* ha facilitato ulteriormente il processo di produzione, aggiungendo anche delle funzionalità alle già tante di *Javascript*, ad esempio nel progetto sono state usate diverse volte chiamate *AJAX* per il recupero dei dati.

jQuery è una libreria JavaScript che nasce con l'obiettivo di semplificare la selezione, la manipolazione, la gestione degli eventi e l'animazione di elementi DOM in pagine HTML, nonché implementare funzionalità AJAX.

### MySQL

*MySQL* è un *Relational Database Management System* (*RDBMS*) formato da un client da utilizzare con terminale a riga di comando ed un server.

Il progetto utilizza una base dati *MySQL*. Per la comunicazione con il server *MySQL* abbiamo utilizzato il *PHP*, che offre diversi metodi per la connessione e l’esecuzione di query.

Inoltre per la gestione delle basi dati, sul server *MySQL*, abbiamo utilizzato il tool in *PHP* *phpMyAdmin*, che ha reso la manipolazione della struttura della base dati e dei dati stessi al suo interno molto semplice ed intuitiva.

### Apache

Come web server locale è stato utilizzato *Apache*.

*Apache* è un server web libero sviluppato dalla *Apache Software Foundation*. È il web server più diffuso ed il suo successo è probabilmente legato alla sua modularità e compatibilità.

È in grado di operare su una grande varietà di sistemi operativi tra cui *Windows* e nel nostro caso *Linux*.

### PHP

Il *PHP* è un linguaggio di scripting interpretato, originariamente concepito per la programmazione di pagine web dinamiche. L'interprete *PHP* è un software libero ed è stato integrato come modulo nel nostro web server *Apache*.

Abbiamo utilizzato il *PHP* principalmente per la parte di logica del progetto. Tutti i controller sono stati scritti in *PHP*. Ma anche nella parte view abbiamo utilizzato il *PHP* come ad esempio il riempimento delle form.

Abbiamo utilizzato un approccio procedurale alla programmazione in *PHP*.

Possiamo definire quindi il *PHP* il pilastro della nostra parte *backend*.

## Organizzazione Base Dati

I dati del progetto sono organizzati in una base dati e di seguito andremo a descrivere dettagliatamente ogni tabella di quest’ultima.

### categories

La tabella *categories* contiene informazioni riguardanti le categorie in cui è divisa la documentazione.

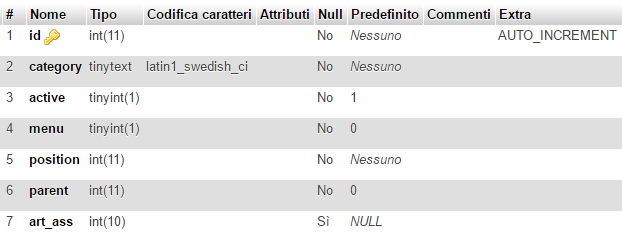


Figura 1 : Tabella “categories”

1. **id:** indice numerico della tabella; si incrementa automaticamente all’inserimento di un nuovo record.
2. **category:** nome della categoria.
3. **active:** booleano che indica se la categoria è attiva o meno; ovvero indica se la categoria verrà visualizzata nel portale destinato all’utente o meno.
4. **menu:** booleano che indica se la categoria è una voce di menu.
5. **position:** intero che indica l’ordinamento tra le varie categorie.
6. **parent:** intero utilizzato per la gestione delle sottocategorie. Se il valore è 0, la categoria in questione è una categoria principale, se il valore è l’id di un’altra categoria, la categoria in questione è una sottocategoria.
7. **art\_ass:** intero utilizzato per legare staticamente un solo articolo alla voce di menù.

### css\_wys

La tabella *css\_wys* contiene le classi *CSS* utilizzate dal plugin *WYSIWYG* create tramite l’apposito tool del software di cui daremo le specifiche complete nelle successive sezioni.

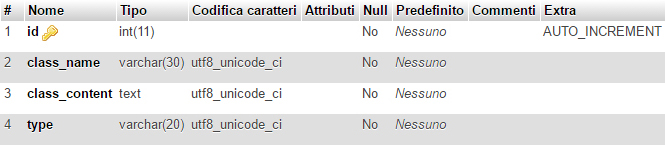


Figura 2: Tabella “css\_wys”

1. **id:** indice numerico della tabella; si incrementa automaticamente all’inserimento di un nuovo record.
2. **class\_name:** nome della classe *CSS*.
3. **class\_content:** corpo della classe *CSS*. Le proprietà sono disposte su una singola riga.
4. **type:** indica il target a cui è rivolta la classe *CSS*. Vi sono 3 possibili valori: text, img, table.

### docs\_it

La tabella *docs\_it* contiene tutti gli articoli della documentazione in lingua italiana.



Figura 3: Tabella “docs\_it”

1. **id:** indice numerico della tabella; si incrementa automaticamente all’inserimento di un nuovo record.
2. **title:** titolo dell’articolo.
3. **date:** data e orario di creazione dell’articolo. Vi è una funzione che riempie automaticamente il campo con i dati del server.
4. **category:** intero che indica l’id della categoria d’appartenenza dell’articolo.
5. **content:** contenuto dell’articolo inviato dal *WYSIWYG*. Sarà composto da *HTML* ed eventuali script *Javascript*.
6. **home:** booleano che indica se l’articolo verrà mostrato come home page del portale utente. Opzione disponibile solo per gli articoli legati alle voci menù.

### languages

La tabella *languages* contiene l’elenco delle lingue in cui è tradotta ed è quindi disponibile la documentazione.

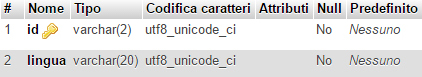


Figura 4: Tabella “languages”

1. **id:** indice numerico della tabella; si incrementa automaticamente all’inserimento di un nuovo record.
2. **lingua:** codice della lingua conforme agli identificatori *ISO-639-1*.

### media

La tabella *media* contiene le immagini caricate sul server ed eventualmente già utilizzate negli articoli.

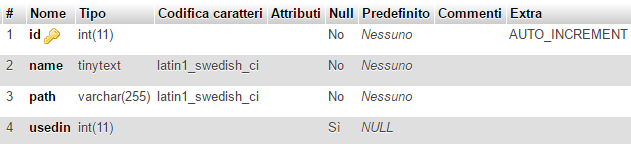


Figura 5: Tabella “media”

1. **id:** indice numerico della tabella; si incrementa automaticamente all’inserimento di un nuovo record.
2. **name:** nome dell’immagine.
3. **path:** percorso relativo dell’immagine.
4. **usedin:** intero che indica l’id dell’articolo in cui l’immagine è eventualmente utilizzata.

### users

La tabella *users* contiene i vari utenti a cui sarà garantito l’accesso al portale.



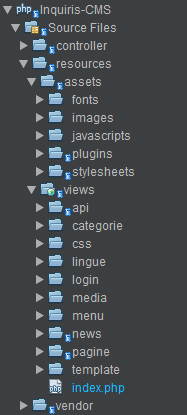
Figura 6: Tabella “users”

1. **user:** username. Chiave primaria in quanto i valori del campo saranno univoci.
2. **password:** password legata allo username.

## Organizzazione del File System

In questo paragrafo discuteremo dell’organizzazione del **file system** del progetto. Vedremo lo scopo di ogni cartella.

Figura 7 : File system progetto



1. **controller:** contiene file *PHP* che rappresentano la logica dietro ogni pagina. Avremo un controller per ogni pagina o quasi.
2. **resources:** contiene tutta la parte di presentazione. Come prevede il modello *MVC*, la parte di presentazione è stata separata dalla quella di logica.
   1. **assets:** contiene tutti i dati utilizzate nelle pagine visualizzate dall’utente e vi sono diverse tipologie di dati. Andiamo ad analizzare queste tipologie nello specifico:
      1. **fonts:** contiene le fonts aggiuntive utilizzare nel progetto.
      2. **images:** contiene le immagini utilizzate nel progetto.
      3. **javascripts:** contiene i gli script JS utilizzati nel progetto.
      4. **plugins:** contiene i plugin utilizzati nel progetto, ad esempio l’editor *WYSIWYG* *“tinyMCE”*.
      5. ***stylesheets:*** contiene i fogli di stile utilizzati *nel progetto.*
   2. **views:** contiene i file *PHP* che rappresentano le varie views, suddivise in sottocartelle.
   3. **vendor:** contiene i plugin installati tramite *composer*.

## Panoramica del progetto



+

Logo Mik\_cms

Tasto per aprire/chiudere il menù laterale

Menù laterale

Titolo sezione

Contenuto della sezione

Figura 8 : Homepage

L’utilizzo di *Bootstrap* ha permesso la realizzazione di una grafica minimale e pulita. I colori caratteristici del portale, che richiamano quelli del logo originale del software, sono il rosso ed il nero.

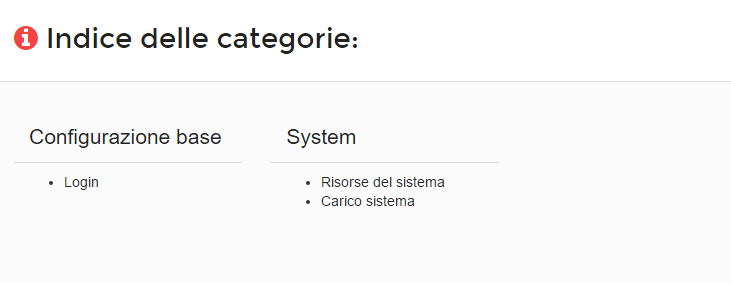
L’*header (1 e 2)*, rappresentato dalla striscia rossa nella parte superiore della pagina, è molto semplice: esso è composto da il logo del software ed un bottone per permettere di mostrare o nascondere il menù laterale.   
  
Il *menù laterale (3)* si divide in un due sezioni: la prima, quella composta dalle voci sino a “*Sezione Live*” rappresenta il modulo **DCMS** dedicato alla produzione di documentazione. Ci occuperemo successivamente di parlare dettagliatamente di ogni voce del menù.

La seconda parte del menù è dedicata alla parte **WCMS**, ovvero a voci per la gestione dei contenuti web del portale dedicato ai clienti.

La sezione dei contenuti *(4 e 5)* è composta da un titolo che permette di tenere sempre traccia della sezione in cui si sta navigando. Il titolo è seguito dalla sezione dedicata al contenuto della pagina.

## Indice documentazione (Homepage)

La sezione contenuto della home è composta da una lista di tutte le categorie della documentazione presenti nella base dati. Possiamo osservare dalla figura 9 come le varie colonne rappresentino le diverse categorie.



Ogni categoria della documentazione è incapsulata in un elemento *box* di *Bootstrap*. Il testo dell’*header* del *box* rappresenta la categoria principale.

Se vi sono eventuali sottocategorie, saranno elencate con un elenco puntato e cliccando su una di

esse, si potrà accedere direttamente alla pagina dedicata alla sua gestione. *Figura 9: Indice delle categorie documentazione*

L’indice delle categorie della documentazione risulta accessibile all’apertura del portale, cliccando sul logo dell’header o tramite la voce “*indice*” della sezione “*Documenti API*” del menù laterale.

Il codice di seguito mostra il controller della pagina *index\_controller.php*:



Codice 1: index\_controller.php

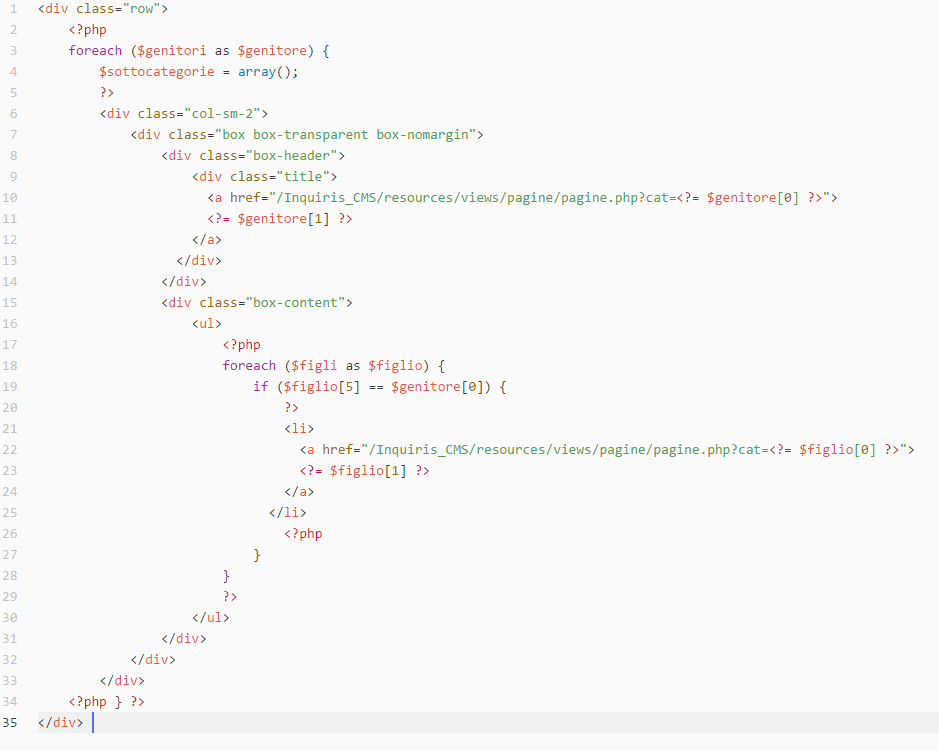
Nella *riga 2* di Codice 1 includiamo uno script php che crea un oggetto *$conn* che rappresenta la connessione al database che sfrutteremo successivamente per eseguire le query e più in generale manipolare i dati della base dati.

La *riga 4* rappresenta la query che preleva dalla base dati tutti i campi, della tabella *categories*, che non sono voci di menù e sono categorie principali, ordinandole in maniera ascendente in base al campo *position*.

La *riga 7* rappresenta la query che preleva dalla base dati tutti i campi, della tabella *categories*, che non sono voci di menù e sono sottocategorie, ordinandole in maniera ascendente in base al campo *position*.

I risultati di queste query vengono inseriti in una struttura dati permanente, un array. Eseguiamo questa operazione in quando i risultati della query possono essere visitati una volta e se non sfruttassimo l’occasione per inserirli in una struttura dati permanente, limiteremmo il loro uso ad una sola scansione. Così facendo, invece, ci riserviamo la possibilità di utilizzare i dati tutti le volte che è necessario. La *riga 20* semplicemente utilizza il metodo dell’oggetto $conn per chiudere la connessione al database.

Analizziamo ora parte del codice *frontend* appartenente alla pagina *index.php*:



Codice 2: index.php

La *riga 3* di Codice 2 rappresenta un *foreach* che visita l’array $genitori creato dal controller *index\_controller.php*. Dell’elemento singolo dell’array $gernitore, che è a sua volta un array che rappresenta una riga della tabella *categories*, utilizziamo l’*id* concatenandolo all’attributo href del tag <a> della riga 10. Alla *riga 11* abbiamo il nome della categoria che visualizzeremo nell’*header* del *box*.

Con la *riga 15* apriamo la sezione *content* del *box* dove con un ulteriore *foreach* sull’array $figli andremo a definire un elenco puntato di sottocategorie, nel caso ve ne fossero.

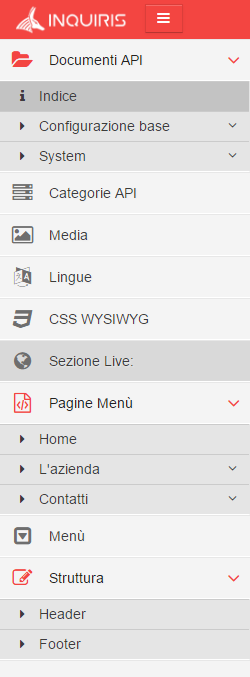
La *riga 19* rappresenta il controllo che verifica se la sottocategoria presa in considerazione è figlia della categoria che il *foreach* esterno sta considerando.

Anche in questo caso, sfruttiamo l’*id* e il *nome* della sottocategoria per utilizzarli come abbiamo fatto con quelli della categoria nelle righe precedenti.

Essendo la piattaforma responsive, il numero dei *box* per ogni riga sarà variabile, mutando in base alla risoluzione e più in generale allo spazio disponibile nell’area in quel momento.

## Menù laterale

Il *menù* è un componente che risiede nel documento *navbar.php* che viene importato in ogni pagina. Come già anticipato, possiamo suddividere il *menù* in 2 aree: la prima è dedicata all’anima **DCMS** del progetto e la seconda a quella **WCMS**.



Nella parte dedicata al **DCMS** troviamo le seguenti aree:

* **Documenti AP**I: elemento del *menù* che raggruppa tutte le categorie in cui è suddivisa la documentazione che produce il software. Come primo elemento statico troviamo l’indice, che ci riporta al contenuto di cui abbiamo parlato nel capitolo precedente. Gli elementi che vediamo nella figura 10, sono creati dinamicamente (“*Configurazione base*”, “*System*”). Sono categorie memorizzate nel database, precedentemente aggiunte tramite il tool che si trova nella voce di menù “*Categorie API*”. Queste categorie possono contenere a loro volta delle sottocategorie e quindi eventualmente avremo un altro livello nel menù.
* **Categoria AP**I: sezione che permette la creazione e manipolazione delle categorie della documentazione.

*Figura 10: Menù laterale*

* **Media:** Gallery che permette di caricare preventivamente delle

immagine sul server e dare a quest’ultime un nome personalizzato.

* **Lingue:** sezione che permette di aggiungere nuove lingue in cui rendere disponibile la documentazione:
* **CSS WYSIWYG:** tool che permette la creazione di classi personalizzate *CSS*, memorizzate nel database e stanziate in un documento che si generà automaticamente prendendo le informazioni dalla banca dati. Queste classi personalizzate

saranno poi utilizzabili dall’editor *WYSIWYG*.

Nella parte dedicata al **WCMS** troviamo invece le seguenti aree:

* **Menù:** sezione in cui è possibile creare e manipolare voci del menù orizzontale presente nel portale dedicato agli utenti.
* **Struttura:** sezione dedicata alla manipolazione dell’*header* e del *footer* del portale dedicato agli utenti.

Il seguente codice è del controller del menù laterale, *navbar\_controller.php*:



Codice 3: navbar\_controller.php

Dalla *riga 3* alla *6* il controller preleva rispettivamente le categorie e le sottocategorie che non sono voci di menù dalla tabella *categories*. Istanzia due array, *righe 7-8*, e successivamente li riempie con i risultati delle query tramite 2 cicli *while*, per rendere i dati prelevati persistenti e iterabili più volte.

*Righe 16-22* troviamo due cicli *foreach* innestati. Il primo cicla sull’array delle sottocategorie $nav\_cat\_figli e l’interno sull’array delle categorie $nav\_cat\_genitori. Confrontiamo l’*id* della categoria attuale (elemento di $nav\_cat\_genitori) ed il campo *parent* della sottocategoria (elemento di $nav\_cat\_figli).

Se coincidono aggiungiamo un campo “*hasChild*” all’elemento corrente dell’array delle categorie per indicare che quella categoria possiede delle sottocategorie. Questa informazione ci servirà successivamente nella parte *View*. Questi dati verranno sfruttati per creare la sezione *“Documenti API”*.

Tutto il processo viene ripetuto nelle *righe 23-43* anche per le voci di menù orizzontale del portale utente ed i dati estrapolati verranno utilizzati per la creazione della sezione *“Menù”.*

Questo è uno snippet di codice della pagina view del menù, *navbar.php*:



Codice 4 : particolare di navbar.php

Alla *riga 2* troviamo un *foreach* che itera sull’array $nav\_cat\_genitori.

Alla *riga 3* abbiamo un controllo sull’elemento preso in considerazione: controlla se la categoria attualmente considerata possiede delle sottocategorie. In base all’esito di questo controllo andremo a definire la struttura dell’elemento del menù.

In particolare, se la categoria contiene sottocategorie, a questo elemento verrà assegnata la classe “*dropdown-collapse*” e sul titolo dell’elemento menù andremo a definire a funzione in *JavaScript* sull’azione “*onClick*”. Questo permetterà sia di esplodere l’elemento del menù cliccando sulla freccia della parte destra, che andare alla gestione degli articoli appartenenti alla categoria cliccando sul nome di questa.

Infine se la categoria non contiene (*righe 31-41*) sottocategorie sarà rappresentata da un elemento menù che effettua il *redirect* cliccando su di esso.

## Categorie API



Figura 11: Categorie API

La sezione *“Categorie API”* permette la creazione e la manipolazione delle categorie in cui sarà suddivisa l’intera documentazione. Abbiamo la possibilità di creare una nuova categoria, creare una nuova sottocategoria, ordinare le categorie/sottocategorie già esistenti, modificarne il nome ed infine disabilitarle/eliminarle.

Ogni categoria (rappresentata da un *box* che varia colore in base allo stato attiva/disattivata) viene inglobata ad un *box* di colore grigio, che conterrà anche ogni sottocategoria ad essa associata.

Il pulsante arancio apre un *box-modal* che permette la modifica del nome della categoria.   
Il pulsante con rosso permette invece l’eliminazione/disattivazione della categoria/sottocategoria.

Il comportamento varia in base a dei controlli: La categoria ha delle sottocategorie? Se si, disattiva invece di eliminare.

La categoria/sottocategoria ha degli articoli di documentazioni associata ad essa? se si, disattiva altrimenti elimina.

Se la categoria/sottocategoria viene disattivata, il colore del suo *box* diventa rosso ed il tasto rosso viene sostituito con uno verde.

L’ordinamento avviene tramite indici interi. Ogni categoria ha la sua casella con l’intero che rappresenta la sua posizione rispetto alle altre. L’intero che rappresenta la posizione viene prelevato dalla banca dati. L’ordinamento è distribuito su due livelli: il primo per le categorie, il secondo per le sottocategorie.

Di seguito il codice del controller della sezione, *categorie\_controller.php*:



Codice 5: categorie\_controller.php

Le *righe 28-45* sono la parte di codice che si occupa di disabilitare/eliminare le categorie/sottocategorie. Il controllo alla *riga 29* verifica che ci siano degli articoli appartenenti alla categoria/sottocategoria che si vuole disabilitare/eliminare.

Alla *riga 32* vi è un controllo che verifica se la categoria in questione è una categoria o una sottocategoria. Nelle restanti righe, sino alla *45*, in base ai controlli effettuati, la categoria in questione sarà eliminata o semplicemente disabilitata.

Le *righe 57-62* sono la parte di codice che si occupa di riabilitare una categoria disattivata in precedenza.

Le *righe 63-72* sono la parte di codice che si occupa della creazione di una nuova categoria. In particolare *riga 64* si occupa di fare l’*escape* dei caratteri speciali del nome della categoria inserito, qualora ce ne fosse il bisogno. Le *righe 65-67* prelevano l’intero che rappresenta la posizione più grande tra tutte le categorie, lo incrementa di 1 e lo inserisce come indice di posizionamento della categoria che stiamo per creare, in modo che risulti ultima nell’ordinamento.

Le *righe 73-83* sono la parte di codice che si occupa della creazione di una nuova sottocategoria e sono analoghe a quelle presenti per la creazione di una nuova categoria.

Le *righe 84-90* sono la parte di codice che si occupa della modifica del nome di una categoria.

Infine le *righe 92-100* sono la parte di codice che si occupa dell’ordinamento delle categorie/sottocategorie. Questa è un’azione innescata modificando l’ordinamento presente tramite il *box-modal*.

Di seguito riportiamo lo snippet di codice che rappresenta un *box-modal* della *view categorie.php*:



Codice 6: particolare box-modal di categorie.php

## Media

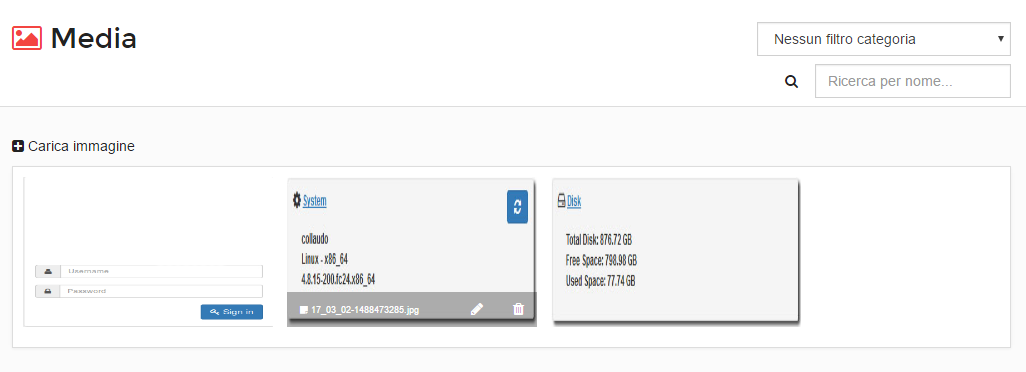


Figura 12: Media

La sezione *“Media”* permette il caricamento preventivo di immagini sul server e le visualizza attraverso una gallery. Abbiamo quindi la possibilità di caricare un’immagine che sia di un’estensione di file permessa e durante l’inserimento possiamo attribuire anche un nome personalizzato a quest’ultima.

Possiamo inoltre modificare l’immagine inserita, sostituendo l’immagine stesso o cambiando semplicemente il suo nome. Possiamo anche eliminare l’immagine e se l’immagine è utilizzata in un articolo, viene avvisato l’utente tramite un messaggio insieme alla conferma di eliminazione.

In alto a destra troviamo altre due funzionalità : *il filtro per categoria* e *la ricerca per nome*.

*Il filtro categoria*, permette di filtrare le immagini visualizzate per categoria, ovvero visualizza solo le immagini utilizzate in articoli della categoria selezionata.

*La ricerca per nome* permette di filtrare le immagini tramite la stringa inserita nella casella di input.

Infine la barra con il nome e le icona di modica e di eliminazione compare quando si ha il mouse su uno dei *box*.

Di seguito lo snippet di codice che rappresenta la gallery:



Codice 7: particolare di media.

Di seguito il codice del controller della sezione, *media\_controller.php*:





Codice 8: media\_controller.php

Alla *riga 19* troviamo dichiarazione della variabile $offset che useremo per l’impaginazione e più precisamente rappresenta il numero massimo di immagini (18) che potranno essere visualizzate in una singola pagina.

Le *righe 20-32* rappresentano il codice che si occupa della gestione della paginazione, del filtro categoria e della ricerca. In base alle varie scelte di questa sezione, si estrapoleranno dati differenti dalla base dati. In particolare le *righe 28-32* si occupano di filtrare in base alla categoria della documentazione selezionata nel menù a tendina ed in questo caso utilizziamo una query con *INNER JOIN* per recuperare i dati.

Le righe 34-61 rappresentano il codice che si occupa del caricamento di una nuova immagine sul server. Le *righe 37-40* si occupano di controllare se l’utente ha digitato un nome personalizzato da dare all’immagine. Se non l’ha fatto, si procede a crearne uno univoco alla *riga 38*.

Le *righe 43-46* effettuano il controllo sull’estensione del file che l’utente sta cercando di caricare e se l’estensione del file risulta non ammessa viene effettuato il *redirect* senza caricare l’immagine visualizzando inoltre un errore che avverte l’utente dell’inconveniente. Se l’estensione è una tra le ammesse viene posta alla *riga 48* la variabile $uploadOk a 1 che ci permetterà al prossimo step di superare il controllo.

Alla *riga 51* si procede all’upload se la variabile $uploadOk è settata ad 1. Alla *riga 52* muoviamo l’immagine nell’apposita cartella del *file system* destinata ad esse. Alla *riga 53* abbiamo la query di inserimento del valore corrispondente nella tabella *“media”.*

Le *righe 75-105* rappresentano il codice che si occupa della modifica di un’immagine. Il codice di questa funzione è molto simile a quello dell’inserimento tranne per le righe iniziali che si occupano dell’eliminazione della precedente immagine, precisamente la *riga 79*. Ed inoltre la query di inserimento diventa una query di modifica alla *riga 97*.

Le *righe 106-115* rappresentano il codice che si occupa dell’eliminazione di un’immagine. L’eliminazione avviene dalla gallery e dal database, senza lasciare alcuna traccia. Se un’immagine è usata in un articolo della documentazione e viene eliminata dalla gallery, non sarà possibile visualizzarla.

Le *righe 116-119* rappresentano il codice che si occupa dell’eliminazione di caratteri speciali e degli spazi bianchi dal nome personalizzato eventualmente inserito dall’utente.

## Documenti API

La sezione *“Documenti API”* permette la creazione e manipolazione di articoli nelle varie categorie della documentazione create in precedenza. Sostanzialmente questa sezione è composta da due livelli.

Il primo si occupa della visualizzazione e dell’eliminazione degli articoli già presenti nella base dati. Il secondo della manipolazione di articoli già esistenti e della creazione di nuovi tramite un editor *WYSIWYG*.

Andremo a parlare dei due livelli singolarmente.

### pagine.php

Questa pagina rappresenta il primo livello della sezione *“Documenti API”* e come già anticipato si occupa della visualizzazione degli articoli già presenti nella base dati.

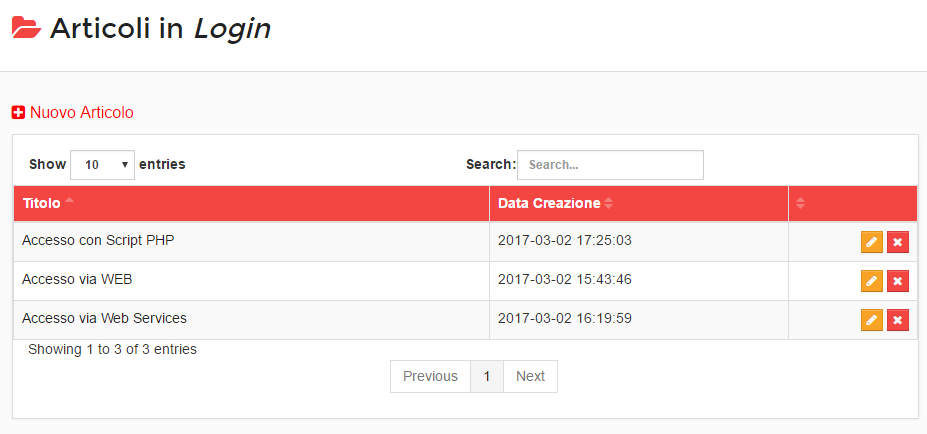


Figura 13: pagine.php

La visualizzazione dei dati presenti nella base dati è delegata al plugin *javascript “datatables”* che si occupa di elaborare i dati e restituirli sotto forma di tabella. La tabella creata dinamicamente offre molteplici funzionalità, tra cui:

* la possibilità di decidere quanti record visualizzare in una sola pagina
* la possibilità di effettuare una ricerca per nome dei record
* la possibilità di ordinamento in base alla colonna scelta
* conteggio totale dei record presenti nella tabella
* paginazione

Oltre alla visualizzazione, abbiamo la possibilità di eliminare, creare o modificare un articolo.

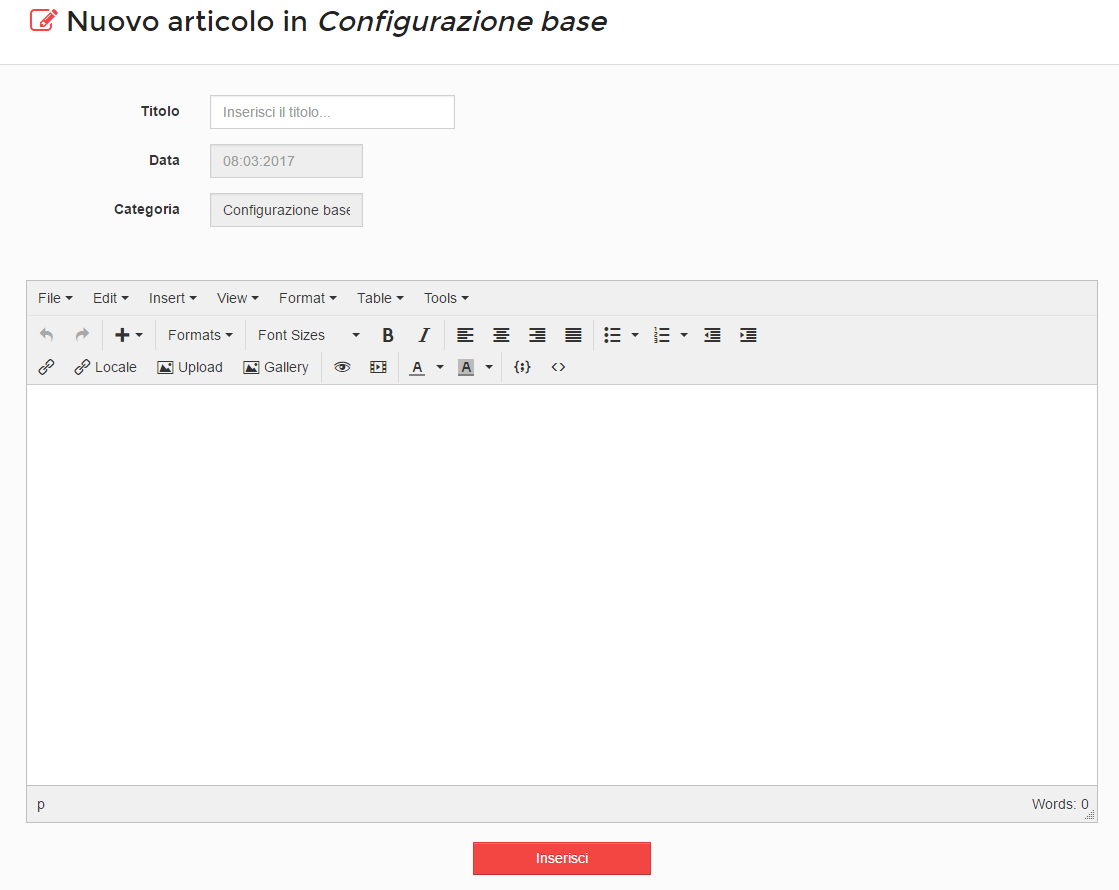
Le ultime due funzionalità sono delegate al secondo livello della sezione *“Documenti API”*.

### articolo.php

Questo livello è il cuore del progetto; ci permette di inserire e modificare articoli della documentazione attraverso un sofisticato editor *WYSIWYG “tinyMCE”* a cui dedicheremo un’intera sezione per descriverne a meglio tutte le sue potenzialità.

La pagina è composta da una form che è composta a sua volta da un elemento che ci permette di inserire il titolo per l’articolo, un campo che si auto-compila con la data odierna, un campo auto-compilante con la categoria a cui apparterrà l’articolo ed infine l’editor *WYSIWYG*.

In caso si tratti della modifica di un articolo già esistente, verrà visualizzato il tipo nella barra *header* della sezione *content* e verrà importato il contenuto dell’articolo nell’editor *WYSIWYG*.

Figura 14: articolo.php

Di seguito il codice del controller *articolo\_controller.php*:



Codice 9: articolo\_controller.php

Le *righe 16-26* rappresentano il codice delegato a controllare se si tratta della modifica di un articolo già esistente, in caso affermativo recuperiamo i dati già presenti nel database per inserirli nella form della view *articolo.php*. In caso negativo, alla *riga 17*, setta la variabile booleana $mod a 0.

Le *righe 29-88* rappresentano il codice delegato all’inserimento dell’articolo nella base dati. Saranno effettuate azioni diverse in base al valore della variabile booleana $mod, ovvero se si tratta della semplice modifica di un articolo già esistente o la creazione di uno ex novo.

Con le *righe 31-34* preleviamo il contenuto dei vari campi della form avendo premura di fare l’*escape* dei caratteri speciali prima di inserire questi dati nella base dati. In particolare $content rappresenta il contenuto dell’editor *WYSIWYG* che può essere composto da codice html e script di altri linguaggi.

Il controllo alla *riga 35* verifica che si tratti di una modifica, in caso negativo ($mod = 0) procede con la sostituzione di una parte della stringa del path delle immagini inserite (*riga 38*). Questo è necessario in quanto, al momento dell’inserimento delle immagini nell’editor *WYSIWYG*, queste ultime verranno inserite in una cartella temporanea e solo all’invio della form verranno spostate nella cartella definitiva destinata alle immagini ed inserite nella base dati.

Dopo queste operazioni preliminari procediamo quindi all’inserimento dell’articolo e delle eventuali immagini nella base dati (*righe 42-47*).

Le *righe 51-87* rappresentano il codice delegato alla modifica di un articolo già esistente ($mod = 1). Effettuiamo anche in questo caso la modifica del path relativo alle immagini da inserire alla riga 52.

Alla *riga 54* troviamo un controllo che verifica che nell’*URL* è stata passata tramite metodo *GET* una variabile “*menu*”. Questa variabile indica che l’articolo in questione di tratta di un articolo legato ad una voce del menù orizzontale del portale utente di cui parleremo nei successivi paragrafi.

Le *righe 67-70* si occupano della modifica di un articolo legato alla documentazione.

Infine le *righe 74-86* si occupano del *redirect* scelto in base al tipo di articolo che si sta inserendo.

Di seguito lo snippet della funzione *“insertImage”*:



Codice 10: snippet insertImage

Alle *righe 96-98* effettuiamo il *parsing* del contenuto della variabile $content alla ricerca di tutti i *tag* *<img>* presenti nel contenuto dell’editor.

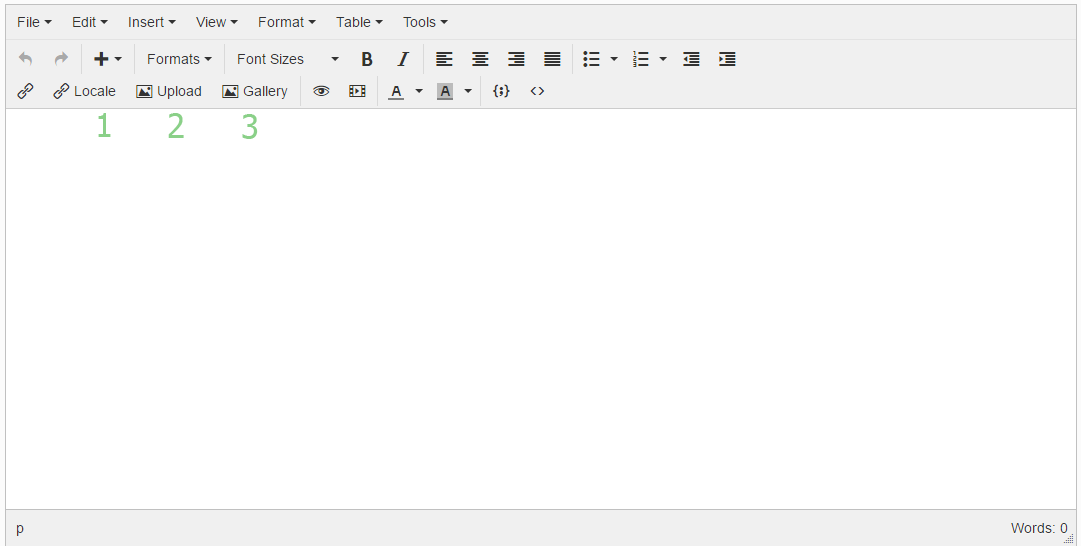
Le *righe 101-113* sono delegate ad inserire tutte le immagini parsate nel database, allo spostamento di queste dalla cartella temporanea a quella definitiva e alla pulizia della cartella temporanea.

## tinyMCE: l’editor WYSIWYG

Dedichiamo un intero paragrafo a questo componente in quanto ne vale la pena descriverne tutte le sue potenzialità ed inoltre sono state effettuate personalizzazioni che vale la pena citare. *WYSIWYG* è l'acronimo che sta per l'inglese *What You See Is What You Get* ("quello che vedi è quello che è" o "ottieni quanto vedi"). Si tratta di un editor incorporato nelle pagine web che permette di modifica le pagine non agendo direttamente sul codice, dando comunque la possibilità di inserire snippet di codice e immagini oltre a semplice testo.

Le funzioni di *tinyMCE* sono le classiche che offre un *WYSIWYG*, ma la vera potenza di questo editor risiede nella possibilità di aggiungere pulsanti con azioni personalizzate e la vasta quantità di plugin che mette a disposizione.

### pulsanti personalizzati

Figura 15: editor WYSIWYG

Per venire incontro alle esigenze della piattaforma sono stati aggiunti 3 pulsanti personalizzati:

1. Pulsante che permette di inserire link ad articoli già presenti nella base dati, generandoli dinamicamente.
2. Pulsante che permette l’inserimento di un’immagine dal file system locale.
3. Pulsante che permette l’inserimento di un’immagine precaricata nella gallery *“Media”* della piattaforma

In particolare il pulsante che permette l’inserimento di un’immagine dal file system locale (2) è stato sviluppato con l’ausilio di un plugin *“JustBoil.me”.*

Il pulsante che permette l’inserimento di un’immagine precaricata nella gallery (3), premendolo mostra un *box-modal* con all’interno l’intera gallery dando la possibilità di inserire l’immagine desiderata.

### plugin “import css”

Plugin che si occupa di eseguire il *parsing* di un file *CSS* che rispetti una precisa sintassi, estrapolandone le classi e redendole disponibili nell’editor.

La creazione del file *CSS* avviene tramite una sezione della nostra piattaforma di cui parleremo in seguito: “CSS WYSIWYG”.

Inoltre come vedremo successivamente tramite lo snippet di codice *javascript* che definisce il nostro editor, abbiamo applicato dei filtri alle classi del file *CSS* parsato, in modo da dividerle in categorie, per rendere la visualizzazione più organizzata e precisamente in:

* **Tabella:** classi applicabili ad elementi *<table>*, *<tr>*, *<td>*.
* **Immagini:** classi applicabili alle immagini.
* **Blocchi:** classi applicabili ai <*div>*.
* **Altri:** classi applicabili al resto degli elementi, ad esempio a *<p>* o <*span>*.

Di seguito lo snippet di codice che definite le funzioni dell’editor nella pagina *articolo.php*:



Codice 11: snippet js tinyMCE in articolo.php

Avremo premura di commentare solo il codice che riguarda i pulsanti personalizzati ed il plugin per la gestione del file *CSS* aggiuntivo.

Le *riga 100* indica il percorso del file *CSS* che il plugin *“import css”* dovrà parsare.

Le *righe 101-106* rappresentano il codice delegato a suddividere le classi trovate dal parse in vari gruppi di appartenenza. Come già anticipato si hanno 4 grandi gruppi: tabella, blocchi, immagine ed altri.

Alla *riga 107* un’interessante opzione per il plugin *“import\_css”: importcss\_merge\_classes* definisce il comportamento del plugin quando deve aggiungere una classe ad un elemento che ne possiede già altre.

Se l’opzione è “*true*” allora il plugin si limiterà ad aggiungere la classe da inserire di fianco quelle già presenti, altrimenti la sostituisce a quelle già presenti.

Alla *riga 108* l’opzione *importcss\_append* permette di visualizzare le classi parsate alla fine del menù a tendina *“Formats”* (vedi figura 15).

Le *righe 110-115* si occupano di aggiungere il bottone personalizzato per l’aggiunta dell’immagine precaricata nella gallery del portale. Il bottone in questione verrà chiamato semplicemente gallery e semplicemente ha su di esso definita una funzione *onclick* che mostra il *box-modal* con *id “imagemodal”* incluso alla *riga 139*.

Nel codice del *modal* troveremo lo script *js* che restituisce all’editor l’immagine selezionata dalla gallery.

Le *righe 117-121* aggiungono il bottone personalizzato per l’aggiunta di un link ad un articolo già presente nella base dati. Anche su questo bottone è definita una funzione *onclick* per mostra il *modal* con *id* “*linkmodal”* dove avremo la possibilità di selezionare attraverso una *datatables* l’articolo di cui vogliamo inserire il link.

Anche in questo caso il codice *js* incaricato di restituire l’id dell’articolo selezionato si trova nella pagina *php* che includiamo alla fine del codice, *riga 140*.

## CSS WYSIWYG

Questa sezione è direttamente collegata con il plugin *“import css”* dell’editor *tinyMCE*. Il tool offre la possibilità di definire delle classi *CSS* e di memorizzarle in modo permanente nella base dati, le stesse classi che saranno usate per generare il file *CSS* che sarà preso in considerazione dal plugin dell’editor.

Vi è quindi la possibilità di creare delle classi *CSS* ex novo o di modificare quelle già create in precedenza.

Le classi CSS possono essere di 3 tipi:

1. **testo**
2. **immagine**
3. **tabella**

Intuitivamente creando una classe per testo, avremo la possibilità di definire delle regole per la formattazione di quest’ultimo.

Creando una classe per immagine, avremo la possibilità di definire proprietà per un’immagine con la possibilità di definirne altre per la pseudo-classe *:hover*, ovvero definire le proprietà dell’immagine quando il puntatore del mouse si troverà su di essa.

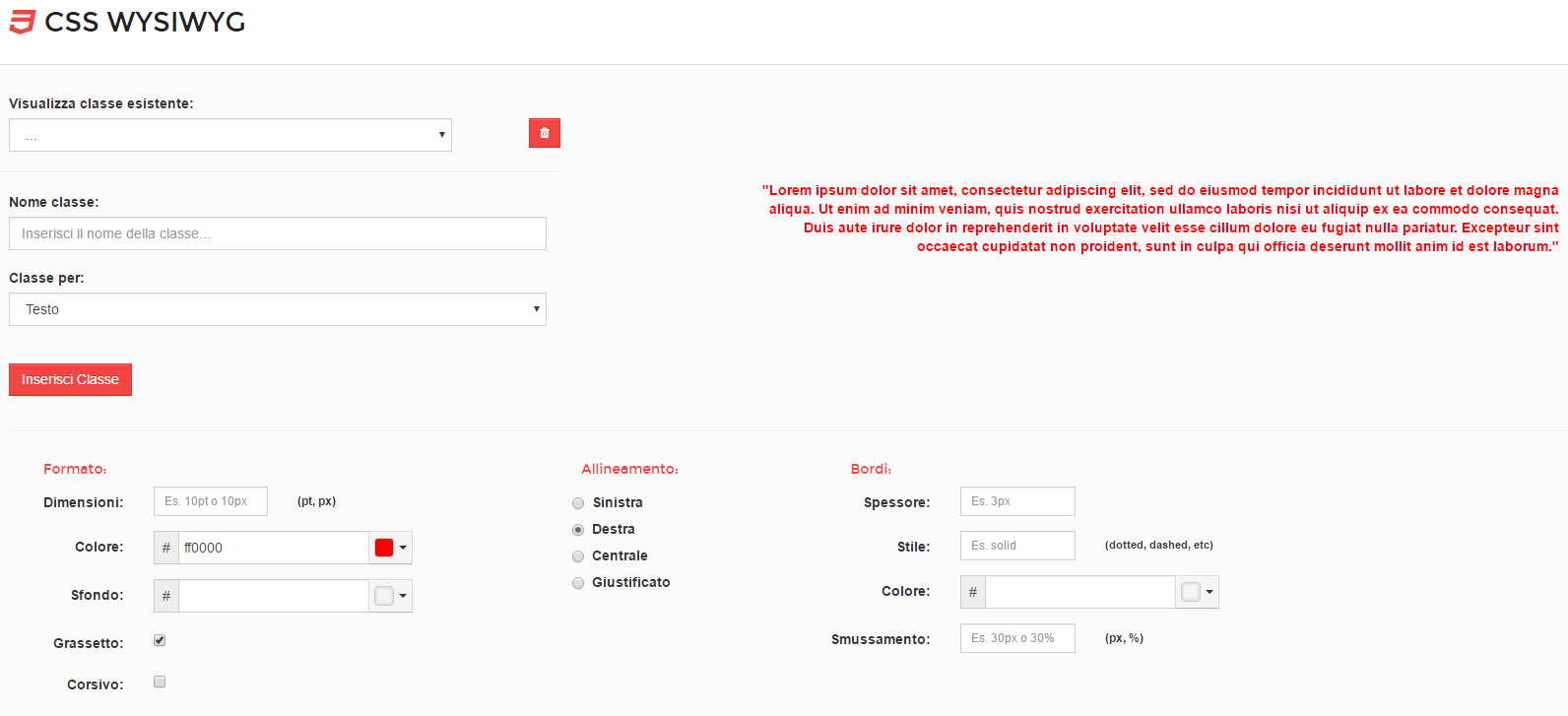
Ed infine la classe per *table*, andrà a definire le varie caratteristiche che daremmo ad una tabella.

Analizziamo prima la parte comune per 3 diverse categoria di classe CSS. Parliamo della form in alto a sinistra che comprende:

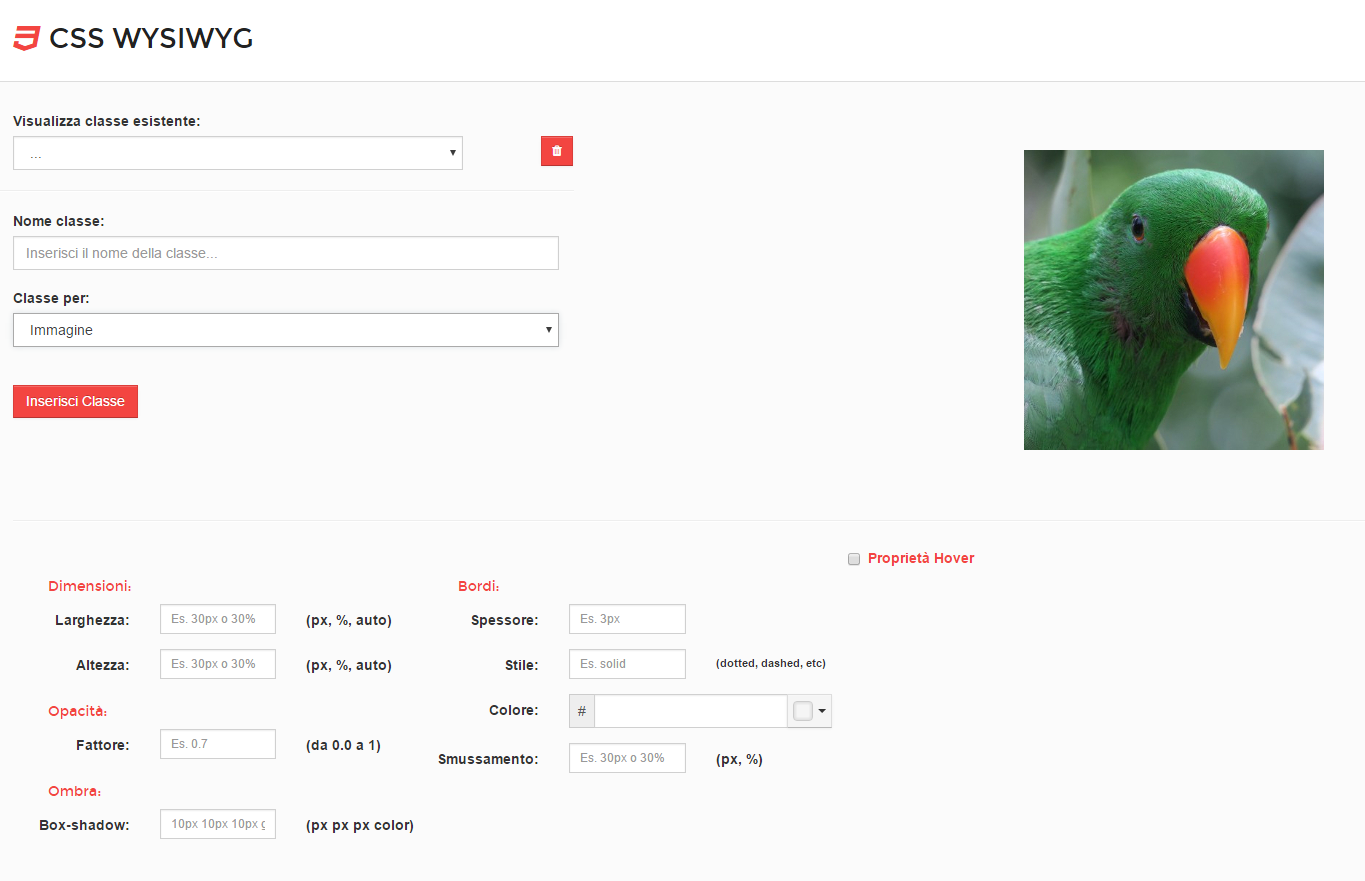
* **Visualizza classe esistente:** è un menù a tendina che permette di selezionare eventuali classi create in precedenza e presenti nella base dati.
* **Pulsante** per l’eliminazione della classe.
* **Nome classe:** è l’input field che permette l’inserimento del nome che daremo alla classe; viene disattivato quando selezioniamo dal menù “**Visualizza classe esistente**” una classe già presente nella base dati visualizzando il nome attuale della classe selezionata. Impediamo di modificare il nome di una classe già esistente perché non avremmo la possibilità di modificare i riferimenti a quella classe negli articoli già creati. Offriamo comunque la possibilità di modificarne le proprietà.
* **Classe per:** menù a tendina che permette di selezionare che tipo di classe andremo a creare. In base a questa scelta, la parte inferiore della form, l’elenco delle proprietà manipolabili cambierà. Inoltre se selezioniamo dal menù “**Visualizza classe esistente**” una categoria già presente nella base dati, questo menù sarà nascosto precludendo all’utente di cambiare la tipologia della classe già creata.
* **Pulsante Inserisci classe**, spedisce i dati della form al controller.

In altro a destra troviamo l’anteprima delle proprietà delle regole già definite.

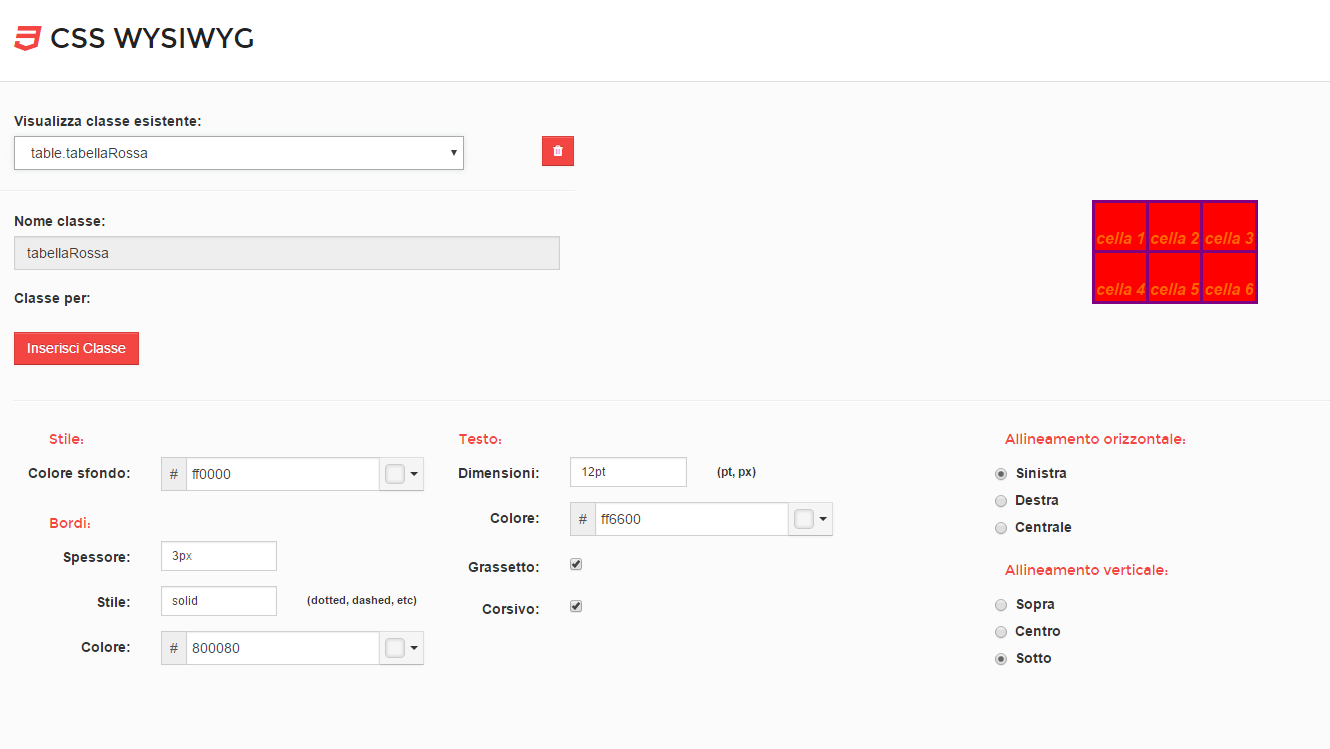
Interfaccia per le gestione di una classe per testo:

Figura 16: interfaccia classe per testo in css.php

Interfaccia per le gestione di una classe per immagine:

Figura 17: interfaccia classe per immagine in css.php

Interfaccia per la gestione di una classe per tabella:

 Figura 18: interfaccia classe per tabella in css.php

Ogni interfaccia del tool chiaramente offre la possibilità di settare proprietà differenti consone alla tipologia di classe che si vuole creare.

Le varie proprietà che definiscono un colore, sono rappresentate da un *input field* con relativo *color picker*, in modo da non dover necessariamente inserire le cifre esadecimali, ma l’utente può semplicemente selezionare il colore del *color picker* e quest’ultimo riporterà automaticamente il codice relativo al colore selezionato nella casella.

Un problema incontrato nella modifica di classi già esistenti è stato il parser. Più precisamente nel momento in cui dobbiamo riempire i campi della form con dati già presenti nella form, c’è bisogno di fare un parsing di tutti i corpi delle varie classi presenti nella base dati.

La fase di parsing avviene nel controller e quindi è *server-side*. Ma i valori restituiti, quelli con cui compilare correttamente la form per visualizzare l’anteprima e permettere all’utente di modificare alcuni valori, vengono gestiti da uno script *javascript* presente nella view *css.php*.

Questo script invia il corpo della classe che si vuole modificare al controller, tramite una chiamata *AJAX* e successivamente interpreta il file *JSON* che viene restituito come risposta dal controller, settando tutti i valori negli opportuni campi della form.

Di seguito lo script che permette il riempimento realtime dei campi della form durante la modifica di

una classe esistente:





Codice 12: script per la gestione form durante modifica in css.php

Alla *riga 138* troviamo una chiamata *AJAX* al controller incaricato di effettuare il parsing del corpo della classe selezionata dal menù a tendina.

Alla *riga 139* andiamo a gestire i dati restituiti dalla chiamata sotto forma di *JSON*.

Effettuiamo il controllo su che tipologia di classe *CSS* si tratta ed in base alla scelta tratteremo il *JSON* in maniera differente.

*Riga 140* controlliamo che la tipologia di classe sia testo, in caso affermativo alle *righe 142-152*, gestiamo i vari elementi dello *JSON*, che in realtà sono le proprietà che compongono il corpo della classe, in maniera differente.

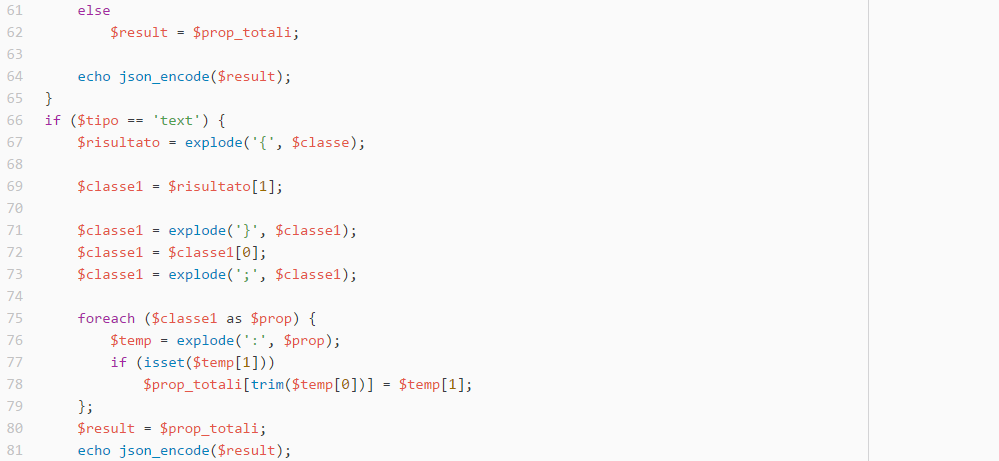
Ad esempio le proprietà rappresentate da *checkbox* nella parte view saranno gestite singolarmente (*riga 144*), cosi come le proprietà rappresentate da *radio button* (*righe 148-149*).

Trattamento esclusivo è riservato anche ai campi con *color-picker* (*riga 151*) ed infine gestiamo anche i semplici *input text* alla *riga 154*.

Le *righe 158 –196* gestiscono i casi in cui la classe sia dedicata ad un’immagine o ad una tabella. Il modo in cui gestiscono il file *JSON* è identico alla gestione di una classe testo.

Codice del controller a cui viene passato il corpo della classe *CSS* da parsare:





Codice 13: pars\_class.php

Il controller *pars\_class.php* si occupa di parsare il contenuto della classe che viene passato tramite stringa e estrapolare le singole coppie proprietà-valore per poi creare un file *JSON* con queste informazioni.

Il corpo della classe viene suddiviso tramite una serie di *explode* fino a raggiungere i singoli valori delle proprietà. È un processo identico in ogni tipo di classe *CSS* che viene passata al controller.

Di seguito analizzeremo il controller che si occupa della creazione del file *CSS* utilizzato dal plugin di *“tinyMCE”*, *css\_controller.php*:





Codice 14: css\_controller.php

Andiamo ad analizzare la parte più interessante del controller, ovvero l’inserimento di una nuova classe *CSS*.

Le *righe 4-5* si occupano di recuperare i dati inviati dal form. In base al valore della variabile $tipo\_classe, andremo ad agire in maniera differente.

Tutte e 3 le scelte operano allo stesso modo, verificano che ci sia una classe con lo stesso nome. Se esiste allora la query sarà di update (*riga 13*) altrimenti sarà di inserimento (*riga 16*).

Le *righe 19-43* si occupano di gestire nello stesso modo le altre due tipologie di classi *CSS*.

Alla *riga 45* viene effettuata una chiamata alla funzione “*create\_css*” che si occupa di creare il file *CSS* ex novo utilizzando i dati della base dati ad ogni inserimento o modifica.

Alle *righe 50-62* troviamo il corpo della funzione *“create\_css”*. Alle righe *51-53* creiamo il file “*wysiwyg\_style.css*” che conterrà tutte le nostre classi *CSS* personalizzate e sarà utilizzato da *tinyMCE*.

Alle righe *54-58* estrapoliamo tutti i copri delle classi dalla base dati e li scriviamo sul file.

Chiudiamo infine lo stream per la creazione del file e gli assegniamo i permessi 777.

## WCMS : Sezione Live

Questa sezione rappresenta il modulo *WCMS* del progetto e ci permetterà di modificare le voci del menù orizzontale che viene visualizzato nel portale dedicato all’utente. Offre la possibilità di creare e manipolare le pagine assegnate ad ognuna di queste voci del menù ed infine potremo modificare l’*header* ed il *footer* del portale dedicato all’utente.

Andremo a parlare in maniera dettagliata di ognuna di queste funzionalità.

### Menù

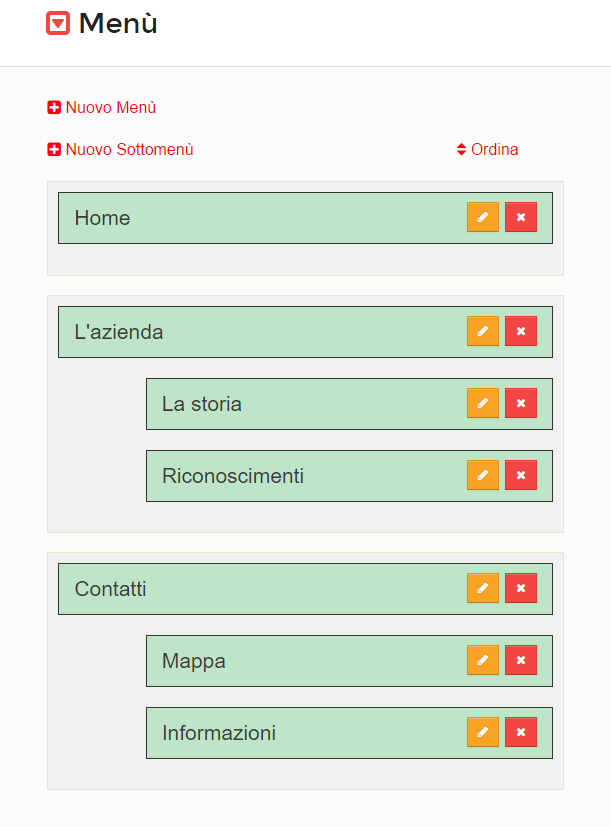


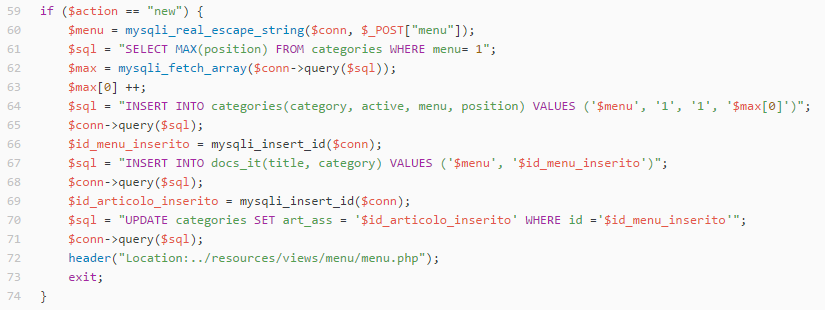
Figura 19 : menù.php

È una sezione del tutto analoga a *“Categoria”* già analizzata nei precedenti paragrafi tranne per un particolare. Al momento della creazione di una nuova voce, viene assegnato a questa nuova voce un articolo in maniera permanente.

Abbiamo quindi la possibilità di creare un nuovo menù ed un nuovo sottomenù; quest’ultimo verrà visualizzato cliccando sulla voce di menù principale.  
  
Tramite editor *WYSIWYG* potremo quindi modificare il contenuto della pagina che verrà successivamente visualizzato nel portale utente cliccando la voce del menù.

È possibile quindi modificare la pagina assegnata ad una voce di menù fintanto che non abbia voci sottomenù assegnata ad essa. Questa scelta è stata effettuata per questioni di logica: fintanto che abbiamo dei sottomenù, cliccando sulla voce di menù principale il menù mostrerà le voce del sottomenù e non l’articolo legato alla voce di menù principale.

Di seguito o snippet di codice tratto da *menu\_controller.php* che si occupa dell’inserimento di una nuova voce menù:

Codice 15: snippet per la creazione voce menù in menu.php

Alla *riga 60* eliminiamo i caratteri speciali del nome postato dall’utente per la nuova voce di menu.

Le *righe 61-65* si occupano qual è l’indice per l’ordinamento più alto già presente tra le voci menù nella base dati, lo preleva incrementandolo di 1 e lo utilizza per l’inserimento della nuova voce menù.

Alla *riga 66* utilizziamo una funzione *php “msqli\_insert\_id”* per prelevare l’*id* autogenerato del record dell’ultima query effettuata.

Alle *righe 67-68* inseriamo quindi l’articolo legato alla voce menù appena creata. L’*id* della voce menù appena inserita prelevato alla *riga 66* sarà utilizzato per il campo “*category*” del nuovo articolo. Il nome dell’articolo sarà il nome della voce menù appena inserita.

Alla *riga 69* preleviamo l’*id* autogenerato dell’articolo appena inserito.

Alle *righe 70-71* andiamo ad aggiornare il campo *“art\_ass”* della voce menù appena inserita, per associare i due elementi: la voce menù e l’articolo.

### Pagine Menù

In questa sezione troviamo un’interfaccia simile alle pagine utilizzate per manipolare gli articoli della documentazione anche se alcuni campi risultato mancanti e ne è stato aggiunto uno.

In particolare nell’interfaccia di questa sezione troviamo una form composta da *input field* che rappresenta il nome dell’articolo ed in questo caso è il nome della voce di menù a cui è associato l’articolo. Il campo risulta non modificabile da parte dell’utente per una questione di coerenza.

Abbiamo una *checkbox* che permetterà di stabilire quale pagina sarà resa Homepage del portale riservato all’utente. Questa caratteristica viene gestita dal controller dal controller *articolo\_controller.php*. Ed infine il classico editor *WYSIWYG “tinyMCE”.*

Di seguito l’interfaccia della *pagina\_menù.php:*

Figura 19: pagina\_menu.ph

### Struttura

In questa sezione abbiamo la possibilità di modificare l’*header* ed il *footer* del portale riservato agli utente. In questo modo vi è una gestione intuitiva e dinamica dei contenuti senza dover mettere modificare direttamente il codice.

L’interfaccia è simili alle precedenti; unica differenza è l’anteprima nella parte superiore della pagina:

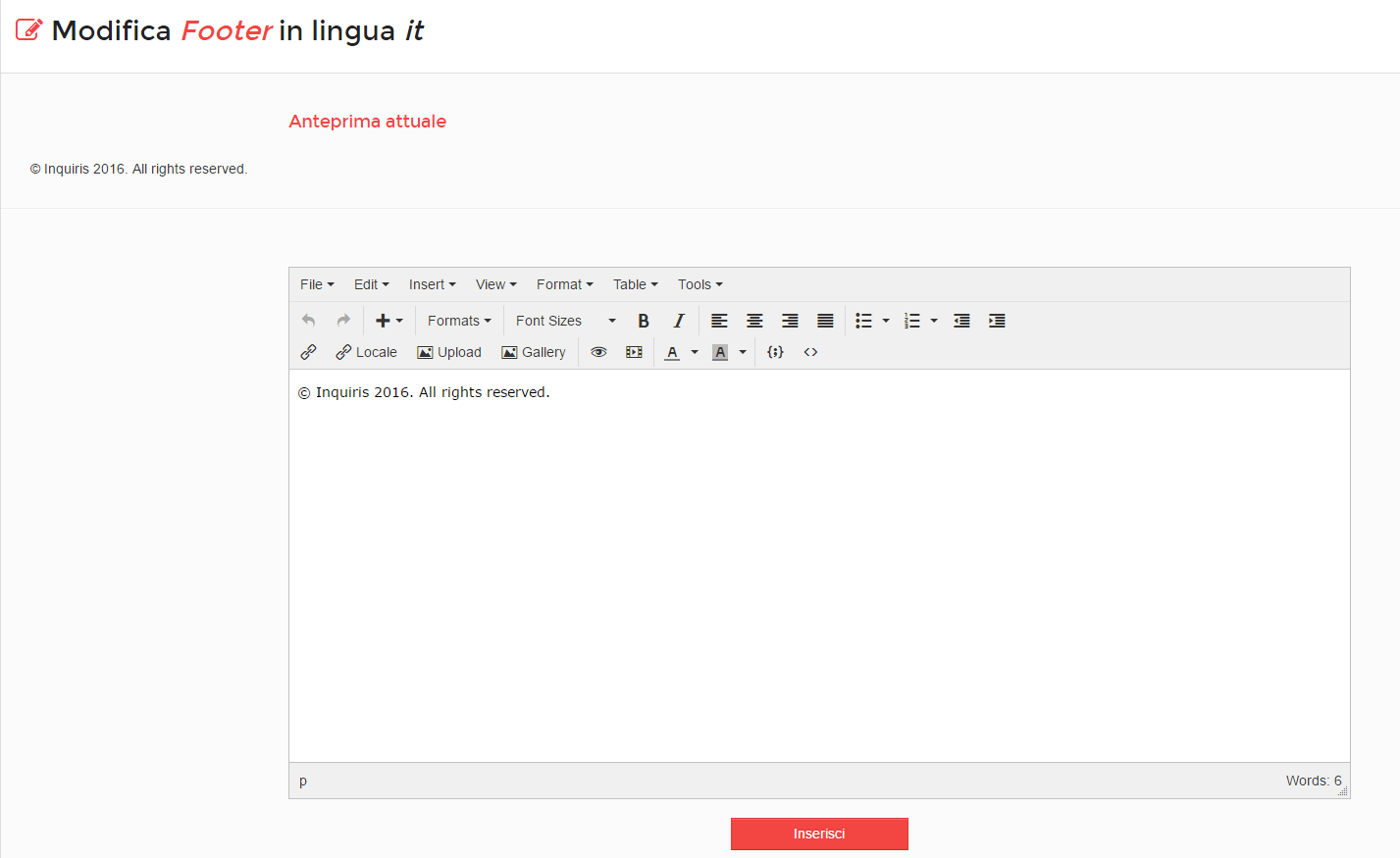


Figura 20: pagina modifica del footer

## Sviluppi futuri: supporto multilingua

Uno dei requisiti necessari affinché un software possa essere distribuito su scala internazionale è sicuramente una documentazione multilingua.

Attualmente non è ancora nata l’esigenza di tradurre l’intera documentazione, ma il progetto è stato sviluppato con una futura predisposizione a questo genere di funzionalità.

In particolare la sezione *“Lingue”* permette di aggiungere delle lingue e tramite il pulsante *“Allinea”* dell’interfaccia tradurre tutte la documentazione già presente in lingua originale nella lingua di destinazione.

Come avverrebbe la traduzione? Tramite le *Google Cloud Translation API.*

L’idea è quella di utilizzare queste API per tradurre il testo dell’articolo nella lingua di destinazione e fare lo storage dell’articolo tradotto nella base dati e più precisamente in *Figura 21: lingue.php*

una tabella creata dell’allineamento tramite l’apposito pulsante, dedicata agli articoli, e quindi alla documentazione, della lingua di destinazione.

Una volta inserita una nuova lingua da supportare, il software ad ogni inserimento di un articolo in lingua originale, tradurrà l’articolo appena inserito in tutte le lingue presenti nell’elenco e stanzierà ogni traduzione nell’apposita tabella.

L’approccio della memorizzazione degli articoli tradotti comporta alcuni svantaggi, ad esempio quello della modifica di un articolo. Infatti la modifica non utilizza un approccio a cascata, questo significa che se nascesse l’esigenza di dover modificare un articolo, dovremmo successivamente modificare quell’articolo manualmente in ogni lingua.

Utilizziamo questa logica in quanto sappiamo che i servizi di traduzione online non sono precisi al 100% e quindi vi era l’esigenza di poter modificare gli articoli per ogni singola lingua per poter correggere eventuali errori di interpretazioni effettuati dal servizio di traduzione Cloud.

Questo è anche uno dei motivi per cui non abbiamo optato per la traduzione *on-the-fly* degli articoli durante la visualizzazione da parte di un utente. L’approccio *on-the-fly* non è stato scelto, a favore dell’approccio statico, per un altro fattore molto importante: quello economico.

Infatti i servizi di traduzione *Cloud*, come quello offerto da *Google*, hanno delle soglie mensili fissate col numero di caratteri tradotti; alla luce di questo, supponendo di avere un discreto traffico sul portale, l’idea di tradurre per ogni singolo utente dall’articolo in lingua originale, risulterebbe sicuramente impraticabile.

Queste logiche sono state già implementate nel progetto. La parte che riserviamo a futuri sviluppi è quella di interfacciamento con le API di Google Traslate.