



APPLICAZIONE WEB PER LA GESTIONE DI UN SERVIZIO DI AUTONOLEGGIO

UNIVERSITÁ POLITECNICA DELLE MARCHE
Tecnologie Web

Studenti:

D'Antonio Angela (1078454)
Nerla Roberta (1080113)
Bernovschi Denis (1077134)
Bolano Khristian (1080185)

Professore

Alessandro Cucchiarelli

15 giugno 2019

Indice

1 Introduzione al progetto	3
1.1 Fasi del Project Development	3
2 Schema of link & Routing	4
3 Descrizione del sito	6
3.1 Header	6
3.2 Pannello Centrale	6
3.3 Footer	6
3.4 Sidebar Menu	6
4 Altri livelli di utenza	7
5 Progettazione del modello relazionale : Concettuale	8
5.1 Entità	8
5.2 Relazioni	8
6 Progettazione del modello relazionale : Logica	10
7 Soluzione adottata	11
7.1 Controllers	11
7.2 Models	12
7.3 Views	12
8 CRUD Operations	13
8.1 CRUD - R - Read	13
8.2 CRUD - C - Create	13
8.3 CRUD - U - Update	14
8.4 CRUD - D - Delete	14
9 Paginator	14
10 Statistics & Prospectus	14
10.1 Implementazione	14
11 Message	15
12 Conclusioni	16

1 Introduzione al progetto

Al fine di approfondire le nostre conoscenze sulle diverse Tecnologie Web, ci è stato assegnato un progetto il cui obiettivo è “*la realizzazione di un sito Web di una società di noleggio auto, che consenta ai clienti di prenotare un’auto vettura tra quelle disponibili per un periodo di tempo definito.*”

La realizzazione è avvenuta tramite l'utilizzo di linguaggi e metalinguaggi sofisticati quali *HTML, CSS, JAVASCRIPT, PHP, JQuery*, con tecnologie di supporto come *PHPMyAdmin, Web Server Apache e l'IDE NetBeans*. La struttura dell'applicazione è basata secondo la logica del framework ”*Zend Framework 1*”. La realizzazione di questo applicativo ci ha permesso di apprendere e sperimentare tutte le tecnologie appena citate, il tutto corredata da una buona dose di curiosità personale.

1.1 Fasi del Project Development

Il lavoro è stato sviluppato secondo tale logica:

1. Discussione progettuale e planning delle attività;
2. Schema dei link e definizione delle pagine;
3. Realizzazione di Mockup e successiva implementazione tramite HTML/CSS;
4. Progettazione del modello relazionale a livello concettuale, successivamente a livello logico ed infine a livello fisico ;
5. Implementazione del database nel Database Management System : ”*MySQL*”;
6. Implementazione graduale dei vari livelli di utenza ¹ dell'applicazione tramite il framework Zend:
 - Area pubblica del sito;
 - Area riservata ai clienti;
 - Area riservata ai membri dello staff;
 - Area riservata all'amministratore.

¹ Livello di utenza : Usufruibilità di un servizio o di un bene a utenti appartenenti ad una determinata categoria. A ciascuna categoria saranno concessi diversi servizi

2 Schema of link & Routing

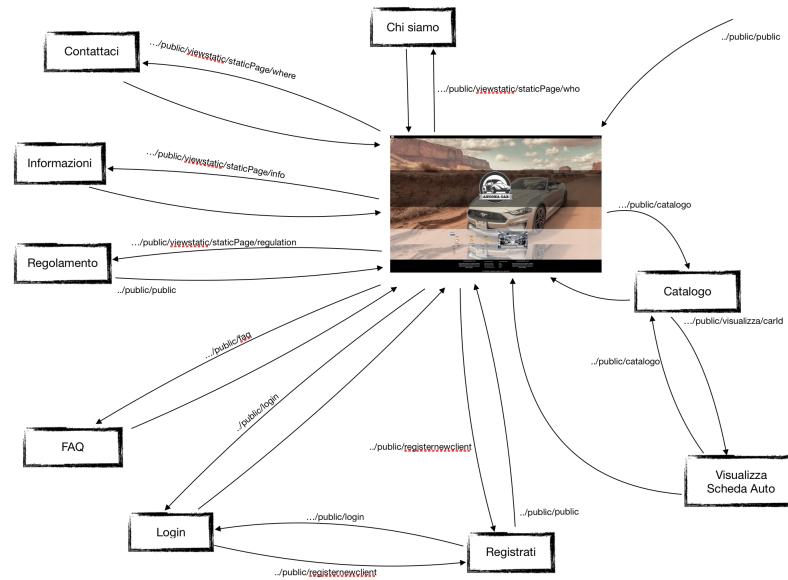


Figura 1: Schema dei link : Public

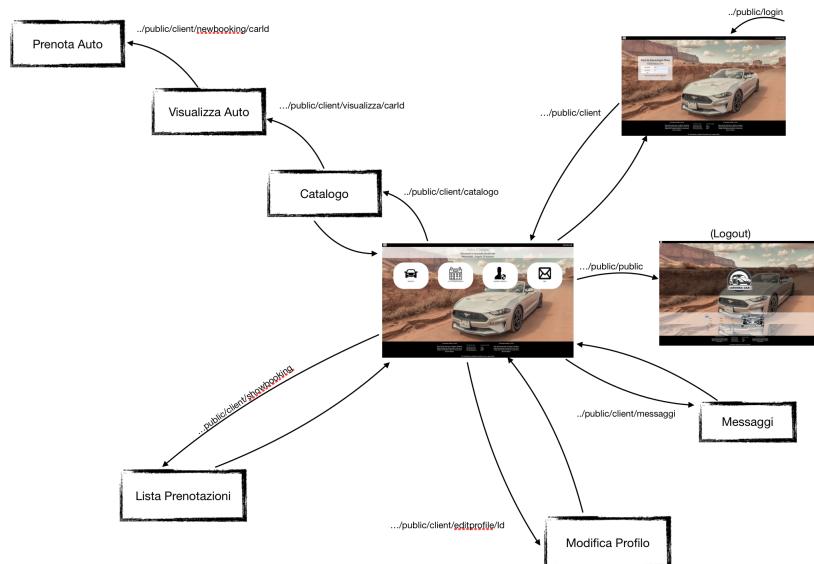


Figura 2: Schema dei link : Client

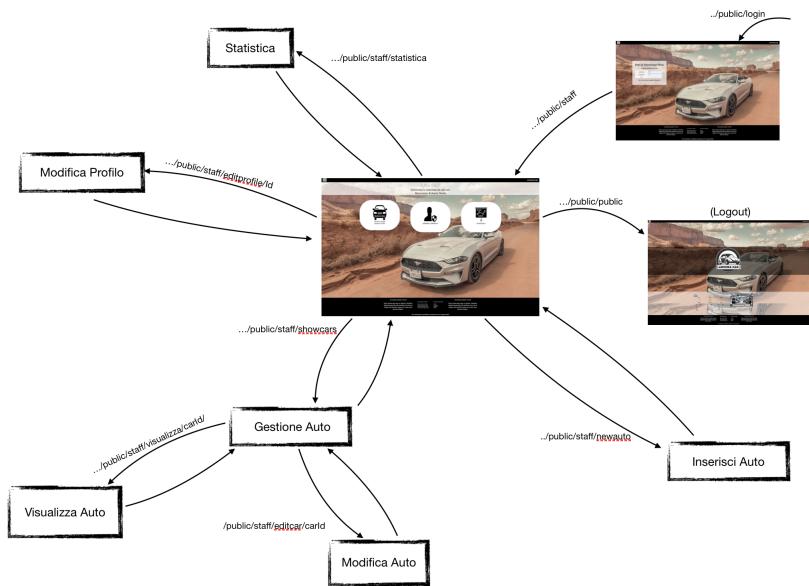


Figura 3: Schema dei link : Staff

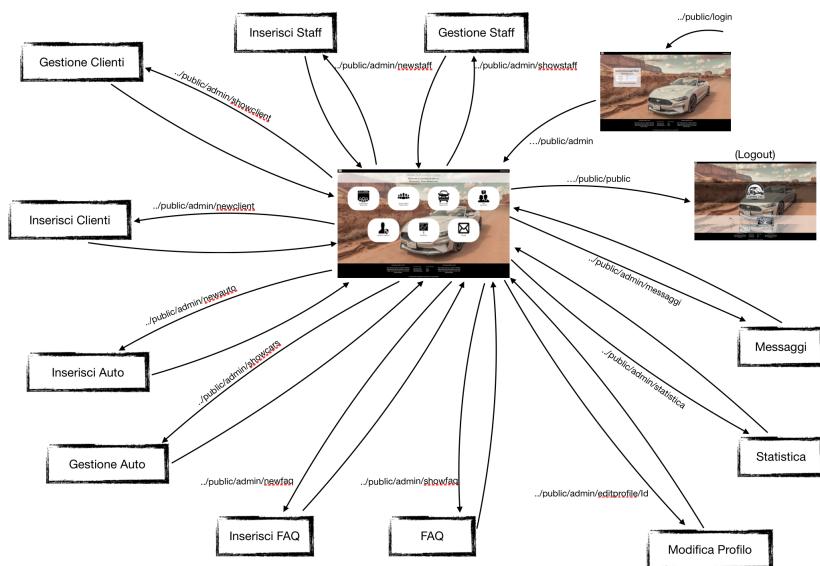


Figura 4: Schema dei link : Admin

Note Per brevità abbiamo optato per non inserire anche il mapping relativo alla modifica nella rappresentazione grafica vista qua sopra, al fine di non aggravare troppo la visualizzazione dello schema.

3 Descrizione del sito

Il sito **AnconaCar** si presenta con una homepage che evidenzia il layout di base dell’intera applicazione web. Ogni singola pagina infatti si presenta con una *Top Navbar*, un *Pannello Centrale* ed un *Footer*.

3.1 Header

L’Header contiene al suo interno due collegamenti a :

- Il tasto 5 una volta premuto apre un ”Sidebar menu”;
- Il tasto 6 se cliccato consente all’utente di tornare alla homepage del sito.



Figura 5: Sidebar



Figura 6: Homepage

3.2 Pannello Centrale

Il pannello centrale è popolato prevalentemente con il contenuto proveniente dalla vista, mostrando in una logica User Friendly, le funzionalità offerte

3.3 Footer

Il footer è dedito invece al contenere informazioni basilari ed alcuni link a pagine esterne all’applicazione web

3.4 Sidebar Menu

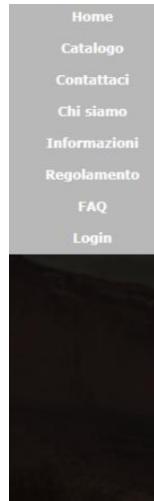


Figura 7: Sidebar menu

Dalla Home Page, attraverso il Sidebar Menu mostrato in figura 7, un utente non loggato ha la possibilità di navigare nel sito cliccando sulle voci di interesse. Come da specifiche del progetto, egli può infatti accedere a diverse aree, tra cui le pagine dedicate al catalogo e alle FAQ. Le voci Login e Registrati permettono all’utente già registrato di loggarsi nell’applicazione oppure dopo avere compilato debitamente una form di registrazione, di

diventare un cliente di AnconaCar.

L'user unregistered può visionare le auto in possesso all'autonoleggio e messe a disposizione, egli ha inoltre la possibilità di filtrare la ricerca a seconda del numero di posti e del prezzo dell'auto come si vede in figura 8.



Figura 8: Auto disponibili e filtri

4 Altri livelli di utenza

Oltre alla sezione pubblica del sito, Il progetto assegnatoci richiede di gestire le diverse classi di utenza: **cliente**, **staff**, **admin**. Ognuno di loro ha a disposizione funzionalità diversificate, può quindi accedere a pagine del sito aggiuntive rispetto a quelle già descritte. Abbiamo dunque deciso che un user registerd verrà immediatamente rimandato alla home a lui dedicata. Da qui e dal sidebar menu egli può accedere a tutte le funzionalità a lui concesse:

- Un **cliente** può effettuare una prenotazione e dunque consultare il catalogo delle auto disponibili, ricercare un'auto applicando dei filtri specifici : *posti, prezzo e date*, gestire le sue prenotazioni, modificare il profilo ed inviare un messaggio all'**admin** del sito, come in figura 9;



Figura 9: Sezione clienti

- Un membro dello **staff** può gestire le auto nel catalogo, modificare il suo profilo e prendere visione delle statistiche riguardanti le prenotazioni, si veda figura 10;
- Infine l'**admin** del sito può gestire lo staff, i clienti, le auto e le FAQ, modificare il profilo, visualizzare il prospetto delle prenotazioni e leggere e rispondere ai messaggi dei clienti. Si veda figura 11.

Ad ognuna di queste funzionalità abbiamo dedicato una pagina specifica, accessibile solo agli utenti che ne hanno l'autorizzazione.

Come già specificato, in ciascuna di esse sarà presente la *top navbar* che permetterà all'utente di, oltre ad aprire il *sidebar menu* ad egli dedicato, accedere alla sezione pubblica del sito e tornare quindi alla *HomePage*. In tal caso nel *sidebar menu* appariranno anche le voci *Logout* e *Nome e Cognome* dell'utente connesso, per consentire a quest'ultimo di raggiungere facilmente la sua homepage personale.



Figura 10: Sezione staff

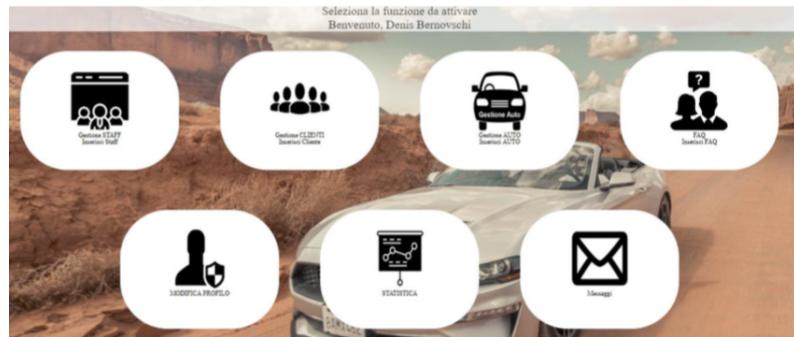


Figura 11: Sezione admin

5 Progettazione del modello relazionale : Concettuale

Per quanto riguarda la progettazione del database, abbiamo iniziato con una sua rappresentazione concettuale attraverso il modello **Entity-Relationship**^{12..}

5.1 Entità

Abbiamo scelto di descrivere i concetti di Utente, Auto e Faq² come delle entità.

Utente Nell’entità *utente* abbiamo optato per la soluzione proposta a lezione di raggruppare gli utenti per gradi utenza, ovvero ad ogni singolo utente è attribuito un **Role** che può assumere i valori : *client*, *staff*, o *admin*.

Auto Nell’entità *Auto* abbiamo espresso i concetti legati all’Auto, quindi in breve abbiamo rappresentato tutte le caratteristiche di una generica automobile.

Faq Nell’entità *Faq* sono rappresentate le informazioni relative alla domanda e alla sua risposta.

5.2 Relazioni

Abbiamo poi completato lo schema attraverso le relazioni **Prenotazione**, **Invio**, **Gestione**:

Prenotazione Coinvolge l’entità *Auto* e l’entità *Utente*, in particolare l’user la cui caratteristica principale è legata a *Role* che deve essere : ”Client”.

Invio Utile all’implementazione della funzionalità opzionale : *Messaggistica* essa infatti permette la relation tra due occorrenze di utente. Le caratteristiche della relationship permettono la gestione del messaggio.

² FAQ : Frequently Asked Questions

Gestione Essa coinvolge l'entità Auto e l'entità Utente in questo caso però l'user ha la peculiarità che deve essere: "Staff" oppure "Admin".

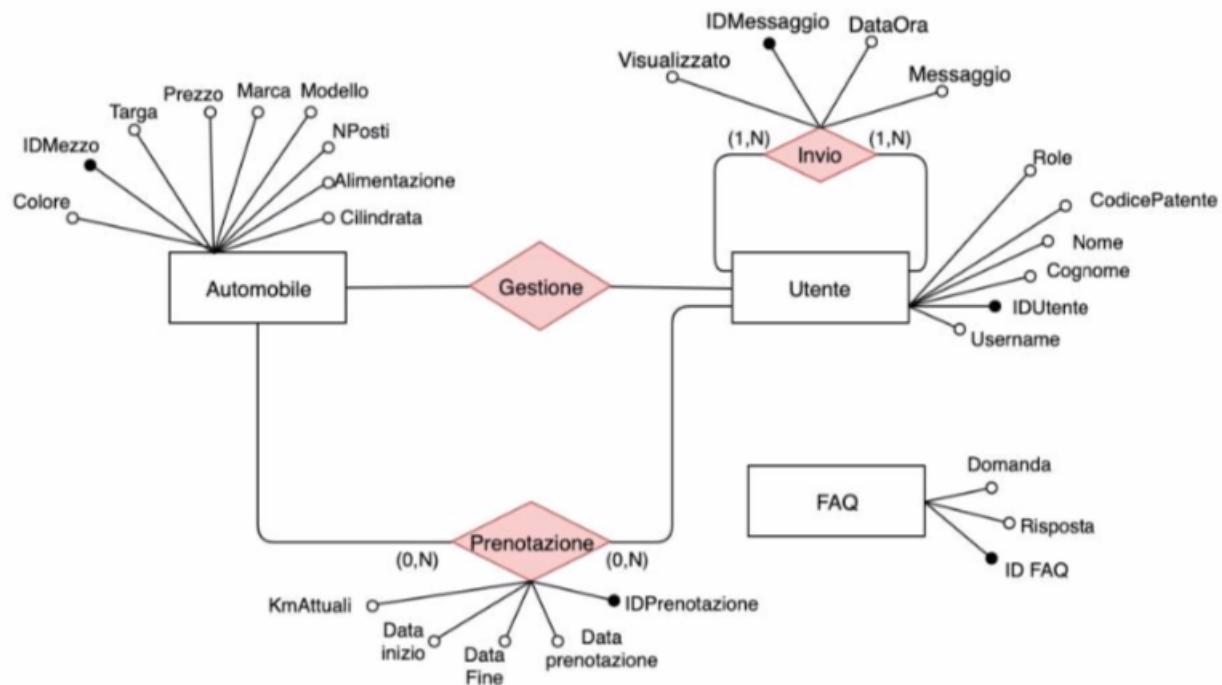


Figura 12: Modello ER

6 Progettazione del modello relazionale : Logica

In seguito ad un'attenta analisi abbiamo tradotto il modello E-R³ in un modello logico funzionale alla rappresentazione dei dati in nostro possesso. Come si può osservare abbiamo eliminato la relationship **Gestione** in quanto essa comportava una ridondanza, infatti, Il costo al fine di mantenerla era nettamente maggiore rispetto al costo che si aveva in caso di eliminazione.



Figura 13: Database Riassuntivo

#	Nome	Tipo	Codifica caratteri	Attributi	Null	Predefinito	Commenti	Extra	Azione
1	IDFaq	int(11)			No	Nessuno		AUTO_INCREMENT	Modifica Elimina Più
2	Domanda	varchar(2500)	utf8_general_ci		No	Nessuno			Modifica Elimina Più
3	Risposta	varchar(2500)	utf8_general_ci		No	Nessuno			Modifica Elimina Più

Seleziona tutto Se selezionati: Mostra Modifica Elimina Primaria Unica Indice Testo completo

Figura 14: Auto

Nome	Tipo	Codifica caratteri	Attributi	Null	Predefinito	Commenti	Extra	Azione
IDMezzo	int(11)			No	Nessuno		AUTO_INCREMENT	Modifica Elimina Più
Targa	varchar(7)	utf8_general_ci		No	Nessuno			Modifica Elimina Più
Colore	varchar(10)	utf8_general_ci		Sì	NULL			Modifica Elimina Più
Marca	varchar(10)	utf8_general_ci		Sì	NULL			Modifica Elimina Più
Modello	varchar(10)	utf8_general_ci		Sì	NULL			Modifica Elimina Più
Alimentazione	varchar(10)	utf8_general_ci		Sì	NULL			Modifica Elimina Più
NPosti	int(11)			No	Nessuno			Modifica Elimina Più
Cilindrata	int(11)			Sì	NULL			Modifica Elimina Più
Under25	tinyint(4)			No	Nessuno			Modifica Elimina Più
Prezzo	double(5,2)			No	Nessuno			Modifica Elimina Più
Image	varchar(30)	utf8_general_ci		Sì	NULL			Modifica Elimina Più
Image2	varchar(30)	utf8_general_ci		Sì	NULL			Modifica Elimina Più
Image3	varchar(30)	utf8_general_ci		Sì	NULL			Modifica Elimina Più

Figura 15: Auto

³ E-R Entity Relationship

Nome	Tipo	Codifica caratteri	Attributi	Null	Predefinito	Commenti	Extra	Azione
IDPrenotazione	int(11)			No	Nessuno		AUTO_INCREMENT	 Modifica  Elimina  Più
IDMezzo	int(11)			No	Nessuno			 Modifica  Elimina  Più
IDUtente	int(11)			No	Nessuno			 Modifica  Elimina  Più
KmAttuali	int(11)			Sì	NULL			 Modifica  Elimina  Più
DataInizio	date			No	Nessuno			 Modifica  Elimina  Più
DataFine	date			No	Nessuno			 Modifica  Elimina  Più
DataPrenotazione	date			No	Nessuno			 Modifica  Elimina  Più

Figura 16: Prenotazione

Nome	Tipo	Codifica caratteri	Attributi	Null	Predefinito	Commenti	Extra	Azione
IDUtente	int(11)			No	Nessuno		AUTO_INCREMENT	 Modifica  Elimina  Più
CodicePatente	varchar(10)	utf8_general_ci		No	Nessuno			 Modifica  Elimina  Più
Nome	varchar(20)	utf8_general_ci		No	Nessuno			 Modifica  Elimina  Più
Cognome	varchar(20)	utf8_general_ci		No	Nessuno			 Modifica  Elimina  Più
Username	varchar(10)	utf8_general_ci		No	Nessuno			 Modifica  Elimina  Più
Password	varchar(15)	utf8_general_ci		No	Nessuno			 Modifica  Elimina  Più
Role	varchar(10)	utf8_general_ci		No	Nessuno			 Modifica  Elimina  Più
Occupazione	varchar(25)	utf8_general_ci		Sì	NULL			 Modifica  Elimina  Più
LuogoResidenza	varchar(20)	utf8_general_ci		Sì	NULL			 Modifica  Elimina  Più
DataNascita	date			Sì	NULL			 Modifica  Elimina  Più

Figura 17: User

7 Soluzione adottata

AnconaCar è stato sviluppato tramite lo **Zend Framework 1**: esso implementa il *Design Pattern MVC*⁴. Seguire un framework, come il sopra citato Zend Framework ci permette di dedicarci completamente al contenuto ed alla logica del applicativo. Inoltre il suo utilizzo ci permette di strutturare meglio l'applicazione, conferendo a quest'ultima un'alta manutenibilità.

7.1 Controllers

L'applicazione predispone di cinque controller, quattro dei quali si riferiscono a diversi livelli di utenza : Public controller, Admin Controller, Staff Controller, Client Controller e infine, l>Error controller, predefinito da Zend Framework. Come **front controller** in automatico Zend Framework prevede venga attivato l'*IndexController*, ma nulla ci impedisce di cambiare tale impostazione nel file *application/config/application.ini*, dove infatti abbiamo definito il Public Controller come *Front Controller*.

⁴ MVC : Model View Control

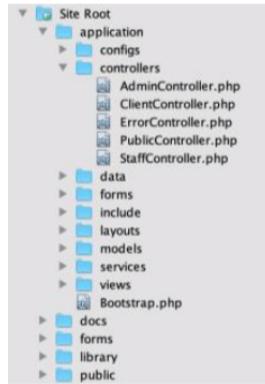


Figura 18: Controllers

7.2 Models

La cartella models contiene un modello astratto per ogni controller definito, in più c'è la cartella resources che include una cartella per ogni tabella del DB, all'interno di queste ultime troviamo un file item e allo stesso livello della cartella resource troveremo la rappresentazione di ogni tabella del DB sotto forma di classe. Questa struttura gerarchica ci permette la definizione di un ORM⁵. Si noti inoltre la presenza del file Acl.php, che permette appunto la definizione di un ACL⁶, ovvero la definizione di regole che determinano l'accesso o meno alle funzionalità e alle risorse in base alla classe di utenza.

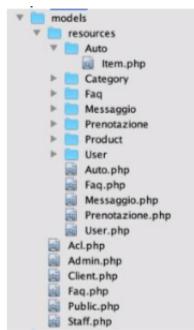


Figura 19: Models

7.3 Views

Si osservi come ad ogni controller, o meglio ancora, ad ogni azione di un generico controller viene definita una vista, la quale a sua volta è collegata gestione dei layout della nostra applicazione.

⁵L'Object-Relational Mapping (ORM) è una tecnica di programmazione che favorisce l'integrazione di sistemi software aderenti al paradigma della programmazione orientata agli oggetti con sistemi RDBMS

⁶ACL : Access Control List

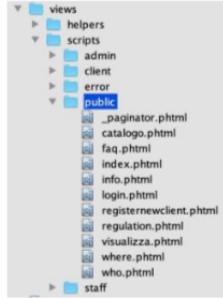


Figura 20: Views

8 CRUD Operations

Passiamo adesso, ad analizzare le azioni più importanti che permettono ai nostri controller di produrre dati. In particolare, essendo l'user **admin** del sito autorizzato a svolgere operazioni CRUD⁷ in diversi ambiti, poniamo l'attenzione sul *AdminController.php*, ed in fattispecie osserviamo la gestione dello staff. Le medesime funzionalità su diverse risorse saranno pressochè simili a quelle qui descritte.

8.1 CRUD - R - Read

La prima azione da analizzare è *showstaffAction()*. Grazie appunto alla funzione *getStaffs()* presente nel file */application/models/Admin.php* ovvero il modello legato al user **admin**. Il modello viene precedentemente istanziato nella function *init()* definita all'interno del *application/controllers/admincontroller.php*. L'azione infatti richiamando il model, che a sua volta richiama il modello associato alla risorsa : *application/models/resources/User.php*, permette di prendere tutte le n-uple della tabella User con l'attributo **Role settato : staff**. Il risultato di tale funzione viene memorizzato in una variabile che verrà poi iniettata nella vista *application/views/scripts/admin/showstaff.phtml*.

8.2 CRUD - C - Create

Qui analizzeremo le azioni legate all'inserimento di un staff member, in particolare porremo l'attenzione alle azioni: *newstaffAction()*, *addstaffAction()* e al metodo *getStaffForm()*. L'azione *newstaffAction()*, consente di generare il viewscript per visualizzare la form di inserimento di un nuovo staff member, nella view viene infatti visualizzata la form ottenuta dal *application/controllers/admincontroller.php*, grazie al metodo *getStaffForm()*. Quest'ultimo ha il compito di istanziare la form definita in *application/forms/Admin/Staff/Add.php*. Una volta ottenuta la form, attraverso un helpers scriviamo l'action da associare alla form. Vedremo poi qui di seguito tecniche di validazione della form lato server. 21 Infine, attraverso le azioni *addstaffAction()* se i dati inseriti sono

```
class Application_Form_Admin_Staff_Add extends App_Form_Abstract
{
    protected $_dateValidator;
    protected $_userValidator;

    public function init()
    {
        $this->_dateValidator = new Zend_Validate_Date ('yyyy-MM-dd');
        $this->_userValidator = new Zend_Validate_Db_NoRecordExists('Utente', 'Username');
```

Figura 21: Admin Staff Add Form

validi, viene richiamata l'azione *insertstaffAction()* o in caso di modifica *savestaffAction()* ripercorrendo la strada *controller → model → resources* permette l'inserimento o la modifica dello staff.

⁷ CRUD è un acronimo, che deriva dalle iniziali dalle quattro operazioni fondamentali compiute nei database: Create (creare i dati), Read o Retrieve (leggere i dati), Update (aggiornare i dati) e Delete o Destroy (eliminare i dati)

8.3 CRUD - U - Update

Le azioni qui analizzate sono `editstaffAction()` che permette la view scripts con la relativa form di modifica `application/forms/Admin/Staff/Add.php` ottenuta tramite il metodo `getStaffEditForm()`. Una volta popolata la form, l'utente può modificare le informazioni da lui desiderate e rinviare la form, quest'ultima verrà verificata, e successivamente come per l'inserimento partirà l'azione di `saveeditstaffAction()` che ripercorrendo la strada `controller → model → resources` salverà le modifiche effettuate.

8.4 CRUD - D - Delete

L'azione qua analizzata è la `deletestaffAction()`. Quest'ultima attraverso il metodo `_getParam()` estrae lo **staffId** associato al staff member che si vuole eliminare. A meno che l'id estratto non sia null, attraverso i metodi `deleteStaffById()` definito nel `application/models/Admin.php` e il metodo `deleteStaffById()` definito questa volta nel file `application/models/User.php`, ovvero quello associato alla risorsa. Viene così cancellato dal database il record relativo a quel staff member.

9 Paginator

Per quanto concerne i **paginator** una volta definita la struttura base grazie al Zend Framework 1 abbiamo modificato il file relativo `application/view/public/_paginator.phtml` al cui interno abbiamo riportato la paginazione come da manuale Zend. Infine osservando che la paginazione era strettamente legata all'implementazione dei filtri *Prezzo, Posti, Data ...* abbiamo optato per la memorizzazione delle informazioni a noi utili in variabili di sessione. Quest'ultime utilizzate in maniera opportuna ci consentono di avere a disposizione le informazioni necessarie al fine di applicare il *paginator*. Per quanto concerne la parte di codice relativo, vi invitiamo ad osservare il codice inserito in : `application/controllers/ClientController.php` e `application/controllers/PublicController.php`. Ci teniamo a precisare che la trattazione dei filtri è stata implementata tramite **Zend Form**

10 Statistics & Prospectus

Le specifiche di progetto richiedono anche che i membri dello staff siano in grado di visualizzare, specificato un mese, l'elenco delle auto noleggiate in quel mese e i clienti che le hanno prenotate. Ci viene richiesto inoltre di mettere a disposizione dell'admin del sito un prospetto in cui, per l'anno corrente, viene indicato il numero complessivo di auto noleggiate per ciascun mese.

Per l'implementazione di entrambi le specifiche abbiamo utilizzato il servizio **Google Chart** che permette la creazione e la personalizzazione di grafici online. Per tale ragione per la visualizzazione del grafico presente nelle view `statistica.phtml` associata all'AdminController e quella associato a StaffController è richiesta la connessione ad Internet.

10.1 Implementazione

In entrambe le viste, le prime righe di codice hanno il compito di caricare le librerie di google chart:

1. Viene caricato il loader stesso di google chart attraverso un file js esterno;
2. La funzione `google.charts.load` della foto carica la versione corrente del servizio ed il pacchetto corechart;
3. Una volta caricato il documento si attiva la funzione di richiamata seguente `google.charts.setOnLoadCallback` che prende come argomento la funzione che si vuole caricare per l'implementazione del grafico. Nel nostro caso `drawChart`.

Analizzando l'implementazione possiamo osservare come in entrambi i controller abbiamo l'azione `statisticaAction()` la quale si occupa di recuperare attraverso il modello e poi attraverso la risorsa, il numero di prenotazioni effettuate. Una volta ottenuto questo valore che per la precisione è un array con dodici posizioni, rappresentante infatti il numero di prenotazione mensile. Questo array passa poi alla vista, e all'interno di `application/views/scripts/staff/statistica.phtml` inizia la sua manipolazione in particolare vengono letti i valori al suo interno al fine di visualizzare ciò che stato richiesto, ovvero le statistiche.

```

<script type="text/javascript" src="https://www.gstatic.com/charts/loader.js"></script>
<script type="text/javascript">

    // Load the Visualization API and the corechart package.
    google.charts.load('current', {'packages': ['corechart']});

    // Set a callback to run when the Google Visualization API is loaded.
    google.charts.setOnLoadCallback(drawChart);

```

Figura 22: Funzione google.charts.load

In dettaglio ... La vista : statistica.phtml associata all’azione, oltre a presentare le righe di codice sopra descritte per il caricamento delle librerie di google chart, definisce, sempre in un tag `<script>`, la funzione `drawChart()` che si occupa di creare e valorizzare la tabella dei dati per la creazione del grafico. Ciò è possibile grazie all’utilizzo di tre variabili:

- La variabile `data` chiama i metodi `addColumn` e `addRows` per costruire la tabella e valorizzare il campo “Numero Noleggi” per ciascun mese sfruttando l’array sopra citato.
- La variabile `options` definisce il titolo del grafico e le sue dimensioni.
- La variabile `chart`, dopo essere stata istanziata, viene utilizzata per chiamare il metodo `draw` che prende come parametri le due variabili sopra descritte. In questo modo viene costruito il grafico ed in particolare viene inserito nel div con Id= `chart_div`.

Per quanto riguarda il codice html abbiamo scelto di inserire oltre al div dedicato al grafico, anche un div per la visualizzazione dei dati richiesti dalla specifica di progetto in un prospetto, nel caso l’utente abbia la difficoltà nel rappresentare il chart. Il grafico che viene presentato nella vista **statistica.phtml** associata allo StaffController è pressoché identico a quello appena descritto, L’azione `statisticaAction` dello StaffController innanzitutto fa quanto descritto per l’AdminController, dopodiché estrae tutte le prenotazioni di un dato mese grazie ai metodi `getNBookingByMonth()` presenti nello staffmodel e nel file di resources `Prenotazione.php`. Bisogna però come nel caso di un staff member avere anche la possibilità di visualizzare le informazioni relative alle auto e ai clienti che hanno effettuato una prenotazione in quel mese.

11 Message

Per il messaging abbiamo optato per una soluzione il più possibile funzionale. L’user client ha infatti nel suo pannello di controllo, la funzionalità di invio messaggio, una volta usufruito della funzionalità troverà un breve riassunto dei suoi messaggi precedenti e la possibilità di inviare un nuovo messaggio. Sarà ora compito dell’admin accedere all’applicativo, una volta loggato avrà anche lui la funzionalità di messaggistica, potrà infatti visionare i messaggi che necessitano un suo intervento e quelli su cui è già intervenuto. Al fine di comprendere meglio la soluzione adottata vi invitiamo ad osservare il codice presente nell’applicativo.

12 Conclusioni

Questo progetto, ci ha permesso di cooperare, apprendere ed applicare tutte le tecnologie viste durante il corso e non, ci ha permesso anche con una buone dose di curiosità di spingerci oltre, di analizzare attraverso attente analisi le problematiche incorse al fine di trovare le soluzioni adatte. Abbiamo discusso e migliorato aspetti fondamentali del progetto più e più volte e dinnanzi a voi troverete la prima versione dell'applicativo. Abbiamo deciso però di non fermarci qua, ma di approfondire ulteriormente le nostre conoscenze, alla ricerca di nuove funzionalità garantendo così un applicativo per quanto possibile user friendly.

Contributo percentuale : D'Antonio Angela (1078454)20%, Nerla Roberta (1080113)20%,
Bernovschi Denis (1077134) 40% , Bolano Khristian (1080185) 20%

Riferimenti bibliografici

- [1] Alessandro Cucchiarelli (2019) - Tecnologie Web - Corso di Tecnologie Web - Università Politecnica delle Marche
- [2] Zend Framework Documentation version : 1.12 - [online] Disponibile a: <https://framework.zend.com/manual/1.12/en/manual.html>
- [3] PHP versione : 5 - [online] Disponibile a : <https://www.php.net/>
- [4] W3C School - [online] Disponibile a : <https://www.w3schools.com/>