



UNIVERSITÀ
DEGLI STUDI
FIRENZE

Scuola di Scienze Matematiche, Fisiche e Naturali
Corso di Laurea in Informatica

Tesi di Laurea

SVILUPPO DI UN PROTOTIPO DI
UN'APPLICAZIONE WEB CON CAKEPHP

DEVELOPING A PROTOTYPE OF A WEB
APPLICATION WITH THE CAKEPHP
FRAMEWORK

BAJRON ISMAILAJ

Relatore: *Lorenzo Bettini*

Anno Accademico 2021-2022

INDICE

1	Introduzione alle tecnologie utilizzate	9
1.1	Breve storia del web	9
1.2	Il linguaggio PHP	9
1.2.1	Scripting di PHP	10
1.2.2	HTTP Embedded	10
1.2.3	Il server PHP	11
1.2.4	Le variabili GET e POST	11
1.3	Apache web server	12
1.4	MySQL	12
1.5	Che cosa è un Framework	13
2	CakePHP	15
2.1	Gli ingredienti di CakePhp	15
2.1.1	Convention over configuration	16
2.1.2	Cakephp e MVC	16
2.1.3	Richiesta in CakePhp	17
2.1.4	Scaffolding e la console	19
2.2	Struttura di CakePHP	19
2.2.1	Model	20
2.2.2	View	21
2.2.3	Controller	22
2.2.4	Middleware	23
2.2.5	Unit Testing	23
3	Realizzazione di un'applicazione web: OrderMe	25
3.1	Breve descrizione dell'applicazione OrderMe	25
3.2	Il modello relazionale	26
3.3	Generazione del codice tramite la console	28
3.3.1	Autenticazione	28
3.3.2	Cake Bake OrderMe	28
3.4	Implementazione dell'applicazione	29
3.4.1	Barra del Menu dell'applicazione	29
3.4.2	Il menu del ristorante	30
3.4.3	Riservare un tavolo in OrderMe con MVC	30
3.4.4	Cart: Il carrello	32
3.4.5	Gli ordini	34
3.4.6	Amministratore e SQL injection	36

3.5	Testing e code coverage	38
3.5.1	Configurazione per il testing	38
3.5.2	Esempi di testing sul menu del ristorante	39
3.5.3	Testing sulla prenotazione del tavolo	40
3.5.4	Testing delle operazioni sul carrello	41
3.5.5	Testing l'azione di invio dell'ordine	43
3.5.6	Code coverage	45

ELENCO DELLE FIGURE

Figura 1	Web service con PHP	11
Figura 2	Interazioni fra Apache, Php e MySQL	12
Figura 3	Interazione tra Php e MySQL	13
Figura 4	L'architettura MVC	17
Figura 5	Ciclo di una richiesta in CakePHP	18
Figura 6	Struttura delle cartelle del framework CakePhp	20
Figura 7	Gestione della richiesta con Middleware	23
Figura 8	Interazioni di base dell'utente nell'applicazione	26
Figura 9	Schema ER del database di OrderMe	27
Figura 10	La barra di menu dell'applicazione OrderMe	29
Figura 11	Le notifiche flash	34
Figura 12	L'insieme dei test effettuati su OrderMe	45
Figura 13	Risultato della copertura dei <i>Controller</i> di OrderMe	46

"A Eleonora e alla mia famiglia"

INTRODUZIONE

Con il susseguirsi di nuove tecnologie e la continua trasformazione di quelle esistenti aumentano le complessità e le difficoltà nella produzione di componenti web di qualità. In tale ambito si inserisce l'oggetto della presente dissertazione che consiste nello sviluppo, la realizzazione e il testing di un'applicazione tramite l'utilizzo del framework CakePHP. Si cercherà, in questo modo, di analizzare il pattern architetturale Model-View-Controller (MVC) e successivamente di implementarlo sul prototipo software OrderMe, che consiste in un gestionale web per la ristorazione. L'obiettivo sarà quello di utilizzare, per lo sviluppo, il paradigma di programmazione orientata (OOP), come fortemente indicato dal framework. Particolare attenzione sarà dedicata alla fase di *testing*, nella quale si tenterà di verificare il corretto funzionamento del codice prodotto. Per concludere, tramite *code coverage*, si valuterà la copertura dei test eseguiti. La trattazione si suddivide nelle seguenti parti: il primo capitolo offrirà una breve introduzione degli elementi dell'architettura tecnologica utilizzata nel progetto, mentre il secondo capitolo descriverà il framework CakePHP; si conosceranno dunque le sue principali caratteristiche e i suoi punti di forza, si analizzeranno i moduli che lo compongono e si scopriranno gli strumenti che esso mette a disposizione per aiutare nella fase di implementazione; il terzo e ultimo capitolo è dedicato al processo di sviluppo e realizzazione dell'applicazione web OrderMe, nel quale si approfondiranno le fasi progettuali del medesimo, come ad esempio la struttura del database. Successivamente, facendo riferimento anche alle parti del codice più significative, si spiegherà la sua implementazione. Chiude la dissertazione la parte del capitolo incentrata sui test, nella quale si valuterà il lavoro svolto tramite alcuni esempi.

INTRODUZIONE ALLE TECNOLOGIE UTILIZZATE

1.1 BREVE STORIA DEL WEB

Con l'avvento di Internet si ha la nascita e lo sviluppo di uno dei suoi servizi principali, il *World Wide Web* (WWW). Nel suo stadio sperimentale, al Cern (*Conseil Européen pour la Recherche Nucléaire*) di Ginevra, Tim Berners-Lee per la necessità della condivisione della grande mole d'informazioni inventa un metodo per il collegamento dei computer conosciuto come HTTP (*Hypertext Transfer Protocol*). Contemporaneamente, utilizzando HTML (*Hypertext Markup Language*) un linguaggio di markup per la formattazione e impaginazione di documenti ipertestuali crea il primo web browser e il primo web *server*.

Si parla quindi dell'architettura *client/server* nell'utilizzo di un servizio sul web. Il *client* esegue, con un browser web sul proprio computer, una richiesta di una pagina web utilizzando il protocollo HTTP, che gestisce la comunicazione al web *server*. Il *server* risponde al *client* con il servizio richiesto e sarà compito del browser di visualizzare la pagina del web, che sarà un documento digitale descritto dal linguaggio HTML. Con lo sviluppo del web, per far fronte alle diverse richieste del *client*, il *server* necessita di interagire con l'utente utilizzando paradigmi di programmazione, da qui la necessità delle pagine web dinamiche.

1.2 IL LINGUAGGIO PHP

Con lo sviluppo dell'architettura *client/server* cresce il bisogno di gestire le pagine web dinamiche; si ha dunque la nascita del linguaggio PHP (*Hypertext Preprocessor*), un linguaggio di *scripting* interpretato e concepito per la programmazione lato *server*. Sviluppato nel 1994 ad opera del danese Rasmus Lerdorf, il quale attribuì ad esso il significato

originale dell'acronimo *Personal Home Page*, si è evoluto negli anni e oggi è considerato uno dei principali linguaggi per lo sviluppo di applicazioni lato *server*. Lo script normalmente viene elaborato dall'interprete PHP, installato sul web *server* il quale produce come risultato una pagina web. Si sbaglia se si pensa al PHP come un linguaggio di *script*, esso non è solo questo, è un linguaggio di programmazione orientata agli oggetti per lo sviluppo di applicazioni complesse Web e mobile.

L'interprete di PHP è distribuito sotto una licenza libera *PHP License* e l'ultima versione rilasciata è la 8.0.1. PHP è un linguaggio di programmazione a tipizzazione debole, dove si può assegnare alla stessa variabile più tipi di dato all'interno della stessa esecuzione del codice. Il tipo di dato, quindi, è associato al valore e non alla variabile. Questo valore che può cambiare dinamicamente in base a manipolazioni esterne.

Troviamo fra le caratteristiche principali del linguaggio:

- *Scripting* di PHP
- *HTTP Embedded*
- Il *server* PHP

Andiamo a vederle più in dettaglio.

1.2.1 *Scripting di PHP*

Nei classici linguaggi di programmazione, per esempio Java o C++, è il compilatore a leggere il file sorgente che contiene le istruzioni per poi produrre gli eseguibili. In PHP i file scritti, detti script, vengono interpretati dall'interprete il quale, leggendo le istruzioni esegue le operazioni ad esse associate.

1.2.2 *HTTP Embedded*

Questa caratteristica si riferisce al fatto che il codice PHP è immerso nell'HTML; gli script sono inseriti, nelle pagine HTML in cui devono produrre i loro effetti. Il web *server* riconosce le pagine PHP, distinguendole da quelle statiche, in base dell'estensione, che non sarà la solita *.html* ma piuttosto *.php*. Quando il *server* riconosce un'estensione associata a PHP passa il testimone all'interprete, lasciando che sia quest'ultimo ad occuparsene.

1.2.3 Il server PHP

Tutta l'elaborazione di uno *script* avviene in modo dinamico sul *server*, prima che questi spedisca la pagina al browser del *client*. Di conseguenza, chi accede ad una pagina PHP non ha la possibilità di leggere le istruzioni in essa contenute, essendo state processate. Ciò che il *client* vedrà sarà il risultato dell'elaborazione. Quindi il *client* vedrà cosa fa lo *script* ma non come lo fa. Questo concetto è molto importante in quanto è alla base della sicurezza e dell'affidabilità offerti dalla programmazione lato *server*, come descritta nella Figura 1.

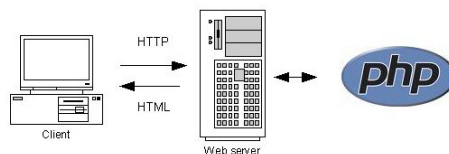


Figura 1: Web service con PHP

Le operazioni lato *server* hanno soprattutto bisogno della possibilità di manipolazione dei dati e PHP è in grado di interfacciarsi a innumerevoli database, offrendo al programmatore il supporto necessario.

1.2.4 Le variabili GET e POST

La principale peculiarità del web dinamico è la possibilità di variare i contenuti delle pagine in base alle richieste degli utenti. Questa possibilità si materializza attraverso meccanismi che permettono agli utenti, sia di richiedere una pagina ad un web server, anche di specificare determinati parametri che saranno poi utilizzati dallo *script* php per selezionare quali contenuti la pagina dovrà mostrare.

Esistono due sistemi per passare dati ad uno *script*: il metodo *GET* e il metodo *POST*.

Il metodo *GET* consiste nell'accodare i dati all'indirizzo della pagina richiesta, facendo seguire il nome della pagina da un punto interrogativo e dalle coppie nome/valore dei dati che ci interessano.

Il metodo *POST* viene utilizzato con i moduli. Quando una pagina HTML contiene un *tag* `<form>`, uno dei suoi attributi è *method*, che può valere *GET* o *POST*. Se il metodo è *GET*, i dati vengono passati nella *query string*, come abbiamo visto prima. Se il metodo è *POST*, i dati vengono

invece inviati in modo tale da non essere direttamente visibili per l'utente, attraverso la richiesta HTTP che il browser invia al *server*.

1.3 APACHE WEB SERVER

Un *server* web è un sistema informatico che si occupa di rispondere a richieste di risorse, dette pagine, usando il protocollo HTTP. Apache *httpd server* è uno dei progetti più importanti della comunità di sviluppatori che lavorano su open source. La figura 2 mostra le interazioni fra Apache, PHP e browser web.

Le pagine servite sono spesso, ma non sempre, documenti HTML che possono includere riferimenti a immagini, fogli di stile e *script*, oltre che testo. In questo caso, il *client* HTTP (ad esempio il browser web) richiede prima la pagina HTML e, dopo averla esaminata, contatterà nuovamente il server per ottenere le risorse esterne per le quali ha trovato i riferimenti URL (*Uniform Resource Locator*). Il *client*, se presente, completerà progressivamente la resa della pagina man mano che tali risorse vengono rese disponibili. Un *server* web può essere utilizzato anche per

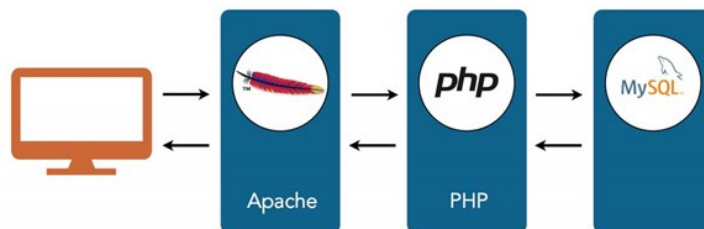


Figura 2: Interazioni fra Apache, Php e MySQL

mettere a disposizione diverse tipologie di file, come archivi compressi ed eseguibili, se il client è in grado di gestirne opportunamente la risposta. Apache supporta il cosiddetto *server-side scripting*, ciò permette di generare dinamicamente i documenti richiesti per poi servirli. Tra i linguaggi di *scripting* supportati il più popolare è PHP.

1.4 MYSQL

Nel mondo dell'informatica esistono differenti *software* utili per la gestione dei database, detti DBMS (*Database management System*). Tra questi ha

riscosso notevole successo anche, molto velocemente, MySQL; un RDBMS (*Relational Database Management System*) *open source* composto da un *client* a riga di comando e un *server*. I punti di forza che hanno permesso a MySQL di raggiungere il successo sono:

- Presenta alta efficienza anche se in presenza di un grande numero di dati;
- Integrazione di tutte le funzionalità che offrono i migliori DBMS: indici, *trigger* e ecc. ;
- Altissima capacità di integrazione con i principali linguaggi di programmazione, ambienti di sviluppo e suite di programmi da ufficio (ODBC, Java, Python, .NET, PHP).

Per utilizzare e amministrare i database MySQL sono stati ideati dei *Manager* appositi fra cui il più importante è phpMyAdmin che permette di creare database e tabelle e di eseguire operazioni di ottimizzazione. Per

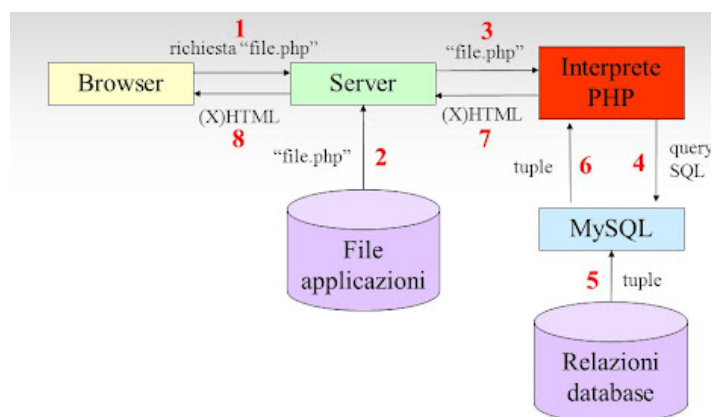


Figura 3: Interazione tra Php e MySQL

l'utilizzo di phpMyAdmin è necessario un web *server* come Apache ed il supporto del linguaggio PHP. Nella figura 3 è rappresentata una tipica interazione *client/server* in presenza di una richiesta di dati utilizzando il database.

1.5 CHE COSA È UN FRAMEWORK

Si può definire un framework come l'insieme di funzioni e *tool* già pronti all'uso, che si possono utilizzare senza doverli progettare da zero

ogni volta. Un framework è un sistema che consente di estendere le funzionalità del linguaggio di programmazione su cui è basato, fornendo allo sviluppatore una struttura coerente ed efficace al fine di effettuare azioni e comandi in modo semplice e veloce.

I componenti principali di un framework sono costituiti da classi collegate tra loro, a cui vengono assegnati dei metodi; tali componenti possono essere statici, ovvero immutabili, oppure flessibili, dunque modellabili. Il framework non è un linguaggio web e nemmeno una libreria di codice, si tratta di una struttura creata per la realizzazione di alcune specifiche funzioni.

Si tratta più che altro di uno dei tanti strumenti che lo sviluppatore ha a sua disposizione per programmare meglio e più rapidamente. Proprio per queste sue caratteristiche, sono diventati sempre più importanti per lo sviluppo di siti web e applicazioni.

Infatti, i framework consentono principalmente di:

- Ottimizzare tempi, costi e benefici offrendo delle librerie e sistemi per assistere lo sviluppatore;
- Evitare al programmatore di riscrivere codice già scritto in precedenza per compiti simili, evitare il problema di reinventare la ruota;
- Organizzare e trovare rapidamente i file necessari consentendo un lavoro in team;
- Concentrarsi sul problema da risolvere, senza dover implementare funzionalità che qualcun altro ha già provveduto a creare.

Per le caratteristiche del linguaggio PHP, la sua comunità ha dato seguito a un grande numero di framework in PHP.

CAKEPHP

2.1 GLI INGREDIENTI DI CAKEPHP

CakePHP è un framework gratuito e *open-source* per lo sviluppo veloce di applicazioni web in PHP, che mette a disposizione tutti gli strumenti di cui ha bisogno il programmatore per implementare le funzionalità progettate. Esso elimina la monotonia e facilita lo sviluppo web, mettendo a disposizione tutti gli strumenti necessari per iniziare a programmare. Potrebbe essere definito come una specie di scheletro iniziale dell'applicazione, che permette di non reinventare la ruota quando si inizia un nuovo progetto.

Le origini di CakePHP risalgono all'anno 2005, quando Michal Tatarynowicz iniziò a scrivere un framework per lo sviluppo rapido di applicazioni in PHP che nominò *Cake*, e successivamente destinato ad una comunità sempre più crescente di sviluppatori. Il nome simpatico *Cake*, la cui traduzione è torta, è uno strumento potente che permette di velocizzare lo sviluppo di applicazioni web, semplificando il modo in cui ci si interfaccia al database.

CakePhp è fortemente mirato alla programmazione orientata agli oggetti, OOP (*Object-Oriented Programming*), in cui le parti dell'applicazione sono delle collezioni di oggetti ed ognuno di essi rappresenta un'istanza di una classe. Tali classi sono legate tramite gerarchie di eredità e le loro rispettive specializzazioni. CakePHP, infatti, supporta i tre meccanismi base della OOP, ossia l'incapsulamento, l'ereditarietà e il polimorfismo. Si utilizzerà la versione stabile CakePHP 4.0.0. dove troviamo i seguenti elementi di forza:

- Utilizza un generatore di codice per lo sviluppo veloce di prototipi di applicazioni;

- Non necessita di configurazioni, tuttavia il database deve essere ben strutturato;
- Il framework viene pubblicato sotto licenza MIT, la quale è molto flessibile;
- Facilita la traduzione, la validazione, l'autenticazione, l'accesso del database e altro ancora;
- Le convenzioni sono semplici e aiutano nello sviluppo;
- Offre un alto livello di sicurezza.

2.1.1 *Convention over configuration*

Convention over configuration è un paradigma di programmazione che prevede una configurazione minima per il programmatore che utilizza un framework che lo rispetta, obbligandolo a configurare solo gli aspetti che si differenziano dalle implementazioni standard o che rispettano particolari convenzioni di denominazione. CakePHP è un tipico esempio di framework che si appropria di questo paradigma.

Imparare le convenzioni per definire l'applicazione costruita con il framework è un processo lungo, che però porta a ottimi risultati dal momento che si ha bisogno di un sforzo minimo per la configurazione. Altri sviluppatori che padroneggiano le convenzioni possono, con un minimo sforzo, partecipare alla realizzazione del progetto. In altre parole, seguendo le convenzioni sarà necessario intervenire sui file di configurazione solo in minima parte. Ciò permette allo sviluppatore di focalizzarsi sulla realizzazione del software e di conseguenza lo sviluppo sarà più rapido. Due esempi classici di convenzioni sono: denominare le tabelle del database con nomi in plurale per esempio *users* invece di *user*, per ogni tabella si ha una chiave primaria che identifica univocamente una tupla (*record*).

2.1.2 *Cakephp e MVC*

CakePHP utilizza l'architettura MVC (*Model, View, Controller*), tradotto con la dicitura modello-vista-controllo; si tratta di un software *design pattern* principalmente utilizzato per la programmazione orientata agli oggetti e in applicazioni web.

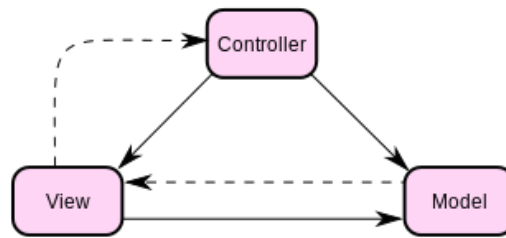


Figura 4: L'architettura MVC

Il *pattern* è basato sulla separazione dei compiti fra i componenti software, ossia i dati, la logica e le regole dell'applicazione:

- *Model* che rappresenta il modello di gestione dei dati dell'applicazione;
- *View* che visualizza la presentazione dei dati e si occupa dell'interazione *client/server*;
- *Controller*, il quale riceve i comandi dell'utente e modifica lo stato delle altre due componenti.

In CakePHP, le tre componenti lavorano in modo indipendente permettendo allo sviluppatore di modificarne uno senza dover cambiare gli altri due. Questo comporta un'applicazione leggera e versatile, poiché nuove caratteristiche potranno essere facilmente aggiunte. In tal modo, sarà possibile costruire uno nuovo sulla base del precedente.

2.1.3 Richiesta in CakePhp

Da un esempio di richiesta - risposta si può sviluppare un'idea della struttura del framework.

Il tipico ciclo di richiesta in CakePHP ha inizio con la richiesta da parte dell'utente di una pagina o di una risorsa dell'applicazione, questa richiesta è inizialmente analizzata da un *dispatcher* (spedizionario), il quale selezionerà il corretto *Controller* che dovrà gestirla. Una volta che la richiesta arriva al *Controller*, questo comunicherà con il *Model* per elaborare i dati che potrebbero essere necessari. Dopo questa comunicazione, il *Controller* delega alla corretta *View* il compito di generare un *output* con i dati forniti dal *Model*. Infine, appena l'*output* viene generato, questo viene immediatamente restituito all'utente.

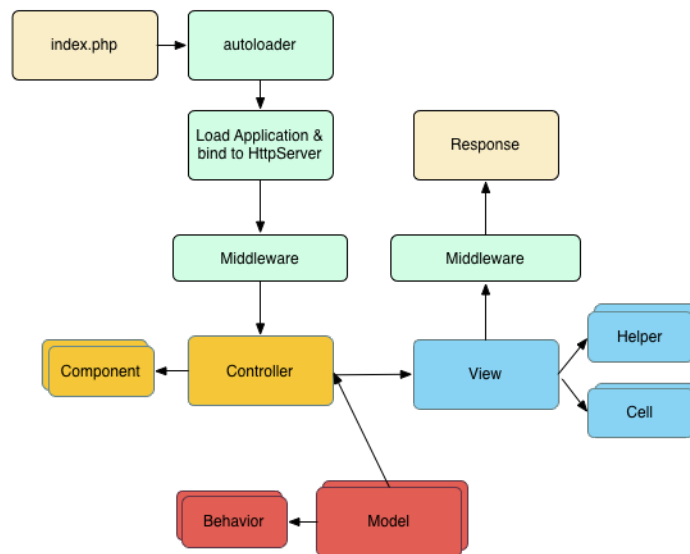


Figura 5: Ciclo di una richiesta in CakePHP

Una tipica richiesta, come descritto nella Figura 5, comincia con l'utente che richiede una pagina o risorsa nell'applicazione attraverso i seguenti punti:

1. Le regole del web *server* reindirizza la richiesta al webroot/`index.php`;
2. L'applicazione viene caricata e associata al *HTTPServer*;
3. Viene inizializzato il *Middleware* (software di mezzo) dell'applicazione;
4. La richiesta e la risposta vengono indirizzate al *Middleware*;
5. Se non si ha una risposta dal *Middleware* un'azione viene selezionata;
6. L'azione su *Controller* viene chiamata per interagire con gli altri modelli;
7. Il *Controller* delega al *View* la risposta per generare l'*output* risultante dal modello dei dati;
8. Il *View* utilizzando gli *Helpers* e i *Cells*, genera la risposta;
9. La risposta passa di nuovo dal *Middleware*;
10. *HttpServer* emette la risposta al web server.

2.1.4 *Scaffolding e la console*

Scaffolding in CakePHP è un metodo che permette allo sviluppatore di definire e realizzare lo scheletro dell'applicazione che implementa le quattro operazioni di base: creare, recuperare, aggiornare ed eliminare gli oggetti. Permette inoltre di definire come gli oggetti sono correlati fra di loro e può creare ed eliminare questi collegamenti. Con *scaffolding* si può creare codice *CRUD* di base di un'applicazione in pochi minuti.

CakePHP, con l'utilizzo della console *Bake*, crea lo scheletro di un'applicazione base funzionante. Partendo dalla struttura del database con la linea di comando `cake bake` si ha la possibilità di preparare tutti gli ingredienti (*view*, *controller*, *model*, *unittest* e *fixtures*) necessari per l'applicazione. Inoltre, tramite la console è possibile avviare un semplice web *server* per le interazioni di base.

Per esempio, se nell'applicazione si vuole rappresentare il concetto di utente e nel database si ha la tabella *users*, allora nella console eseguendo il comando:

```
bin/cake bake all users
```

e si ha la creazione dei file che rappresentano: *Table*, *Entity*, *Controller*, *CRUD templates* e i tests rispettivi a la tabella *users*. Questo è un strumento potente per la generazione automatica di file in modo tale da fornire al programmatore una struttura da cui cominciare subito a raffinare l'applicazione che desidera costruire.

2.2 STRUTTURA DI CAKEPHP

Una volta installato CakePHP la cartella avrà il contenuto descritto nella figura 6:

Fra le principali cartelle si ha:

- **config:** Contiene file di configurazione come i connettori con il database, i file d'accesso e i file di sicurezza;
- **src:** Qui si trovano i file che rappresentano il cuore dell'applicazione che si sviluppa ed è possibile trovare le componenti principali come i *Controller*, *Model* e i *View*. Contiene inoltre i comandi e i *Middleware* per il *routing*;

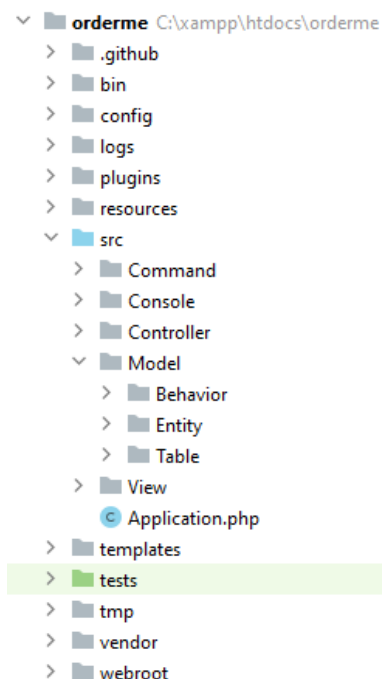


Figura 6: Struttura delle cartelle del framework CakePhp

- **templates:** L'utente esterno comunica con l'applicazione tramite i file che si trovano in questa cartella;
- **vendor:** Contiene terze parti di codice ma soprattutto le varie librerie che sono la base per il funzionamento del framework;
- **tests:** I file e le cartelle necessarie per eseguire i test;
- **webroot:** Contiene i file pubblici dell'applicazione pubblici come gli stili, le immagini o i file di javascript utilizzati.

2.2.1 Model

Il *Model* è il primo concetto dell'architettura MVC. La comunicazione con gli archivi dei dati come le tabelle, i file strutturati o i record sono aspetti fondamentali di un'applicazione web, specialmente se coinvolge un grande numero di utenti. L'azione di manipolazione dei dati memorizzati in database conviene gestirli tramite il *Model*.

Il *Model* è responsabile dell'interrogazione del database per la lettura dei dati e della loro conversione nei concetti base dell'applicazione. Più

precisamente include l'elaborazione, la convalida e l'associazione dei dati. In CakePHP questo concetto è descritto tramite le componenti *Table* che rappresenta la tabella del database e *Entity* che corrisponde ai singoli record nella tabella.

L'aspetto di programmazione agli oggetti di CakePHP si distingue specialmente nel rappresentare i dati del database in oggetti, cioè obbliga il programmatore all'utilizzo della tecnica di programmazione ORM (*Object-relational mapping*). Tramite questa tecnica si implementa l'uso dell'interfaccia di *Model* per tutti le operazioni inerenti agli dati che saranno rappresentati come oggetti del framework in tal modo da facilitare il processo di sviluppo.

Tornando all'esempio della tabella *users*; lo *scaffolding* di CakePHP, genera due classi in PHP che descrivono il modello (*Model*); la classe *UsersTable.php* che rappresenta la tabella nel database, e la classe *User.php* che corrisponde al singolo record della tabella.

ORM di CakePHP offre i seguenti metodi che operano sul database per le classiche operazioni di manipolazione dei dati:

- `get()`: Selezione della tabella di database, passato come attributo, su qui effettuare le operazioni;
- `find()`: Metodo utilizzato per l'interrogazione del database;
- `save()`: Aggiornare/aggiungere una tupla nella tabella;
- `delete()`: Eliminazione del record.

La validazione è implementata all'interno della classe '*Table*'. Infatti, ci sono i relativi metodi che consentono di filtrare i dati ricevuti dall'utente prima di passarli al database. Sono presenti metodi per effettuare i controlli più comuni sui dati quali ad esempio, la lunghezza, la verifica dei record per validare le email e così via.

2.2.2 View

View è responsabile per la presentazione di un specifico risultato richiesto dal utente. CakePHP offre una grande varietà di classi che sono utili a gestire la maggior parte degli scenari possibili nella presentazione dei dati all'utente.

Sicuramente quelle più significative sono i *templates*, posizionati nella

cartella omonima. Il generatore di codice di CakePHP genera automaticamente una classe per ogni azione visibile contenuta nei *Controller*, classe che si occupa della presentazione dei dati come risposta dell'azione intrapresa dall'utente.

Tornando all'esempio della tabella *users*, il generatore crea dentro la cartella *templates* la cartella *Users* che contiene i file *add.php*, *edit.php*, *view.php* e *index.php*, i quali rappresentano le quattro azioni di base quando un utente interagisce con l'applicazione in merito ad un oggetto, in questo caso *users*.

Gli *Helpers* sono delle classi incapsulano la logica nel caso in cui vi sia necessità di una presentazione più raffinata.

Qualora sia necessario di presentare all'utente dei dati interni, sarà compito del *Controller* che tramite il metodo *set()* passa come argomento il dato richiesto al *View*, il quale tramite i *templates* lo presenta all'utente.

2.2.3 Controller

Il *Controller* ha la responsabilità di gestire la parte logica dell'applicazione. I metodi in esso contenuti rappresentano le azioni dell'applicazione e ogni metodo rappresenta una singola azione.

Un tipico comportamento di un *Controller* è il seguente; quando arriva una richiesta, interroga il *Model* per poi elaborare i dati e infine selezionare il *View* desiderato. Quest'ultimo si occupa di presentarli all'utente. Si può pensare al *Controller* come un strato intermedio tra il *Model* e *View* dove si occupa della parte logica per gestire i dati generati dal *Model*.

I metodi pubblici del *Controller*, cioè le azioni, sono responsabili per convertire la richiesta di dati in una risposta del browser utilizzando le convenzioni automatizzate per questo processo che altrimenti risulterebbe laborioso.

CakePHP richiama le azioni di un determinato controller in base alle richieste provenienti dal URL. Per esempio se nella classe *UserController* si trova il metodo *add()* che rappresenta un'azione allora si può interagire con il *Controller* tramite l'indirizzo web `https://.../users/add/`.

2.2.4 *Middleware*

Un *Middleware* funziona da involucro per l'applicazione, con lo scopo di gestire determinate richieste e generare le relative risposte. È possibile vedere l'applicazione avvolta dagli strati del *Middleware*, schematizzata nella seguente figura 7:

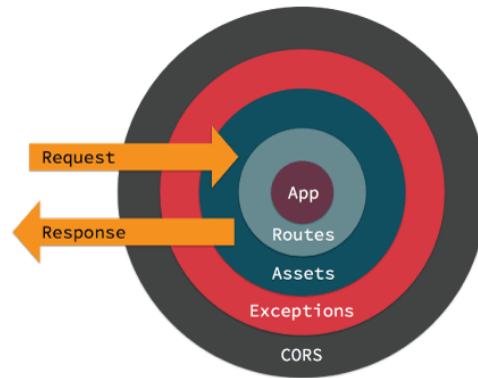


Figura 7: Gestione della richiesta con Middleware

Il *Middleware* che gestisce le richieste prima che arrivino all'applicazione per esempio è incaricato di applicare il percorso corretto delegando il compito al *Controller* specifico su cui è indirizzata la richiesta oppure il reindirizzamento in caso di eccezioni.

2.2.5 *Unit Testing*

Per *Unit testing* si intende l'attività di *testing* di singole unità software dove unità normalmente è intesa come il minimo componente di un programma. A seconda del paradigma o linguaggio di programmazione, questa corrisponde per esempio a una singola funzione nella programmazione procedurale, una singola classe o un singolo metodo nella programmazione a oggetti.

CakePHP è fornito con un supporto completo per i test integrato con il framework per il *testing*, PHPUnit, che è lo standard per i test d'unità in PHP. Oltre alle funzionalità offerte da PHPUnit, CakePHP offre delle altre aggiuntive per semplificare il processo di *testing*. Ad esempio, CakePHP utilizza PHPUnit come framework di test sottostante. Offre un set completo di strumenti con delle funzionalità per assicurare lo sviluppatore che il codice esegui ciò che pensa. Dividendo l'applicazione

sotto test in piccole classi, chiamandosi *Test Case*, in modo tale da testarle separatamente.

I metodi rilevanti sono:

- `setUp()`: Chiamato prima di ogni metodo di test per creare l'oggetto che si deve testare;
- `tearDown()`: Chiamato dopo la fine del test per distruggere l'oggetto testato.

Tramite PHPUnit si è in grado di generare anche *Code Coverage* (copertura del codice) che rappresenta la percentuale della linea di codice coperta dall'esecuzione che ha passato il test positivamente. *Code Coverage* è una metrica della quantità di *UnitTest* eseguiti, e compito di quest'ultimo dare buone garanzie sul corretto funzionamento del codice.

Fixtures

Le *fixtures* consentono di proteggere ed isolare i test attraverso la creazione di dati predefiniti. Utilizzando questi dati sarà semplice testare il comportamento dell'applicazione in modo tale da evitare l'uso del database. Le *fixtures* sono create in automatico con il *Bake* dei modelli e sono inserite nella cartella `/Test/Fixture`.

Tutte le *fixtures* estendono la classe `CakeTestFixture` dove il campo pubblico `$fields` contiene la definizione dei campi e rispecchia dunque la struttura della tabella facilitando il processo di *testing*.

REALIZZAZIONE DI UN'APPLICAZIONE WEB: ORDERME

3.1 BREVE DESCRIZIONE DELL'APPLICAZIONE ORDERME

OrderMe è un semplice prototipo di applicazione web sviluppato per comprendere il framework CakePHP e per eseguire un processo di *testing* su le sue funzionalità.

Per lo sviluppo dell'applicazione si fa uso della piattaforma web server XAMPP che include Apache, MySQL e PHP. Come IDE (*Integrated development environment*) si utilizza PHPStorm che può essere sostituita con qualsiasi altro *editor*. Si utilizza CakePHP versione 4.0.0 con il suo stile CSS di base e per sviluppare il codice si utilizza la lingua inglese.

OrderMe è un gestionale web per la ristorazione, tradotto 'ordinami', gestisce l'interazione di un commensale con il cameriere di un ristorante. Si può descrivere il ruolo di OrderMe come un cameriere virtuale.

Il commensale, da qui in poi denominato utente normale, collegandosi al sito del ristorante può visualizzare il contenuto del menu. Se decide di usufruire dei servizi di ristoro, cioè è fisicamente presente nel ristorante, seleziona un tavolo fra quelli disponibili da riservare e a questo punto l'utente può procedere con l'ordinazione. L'ordine consiste in due fasi: inizialmente si aggiungono i piatti desiderati al carrello virtuale e successivamente si procede con la convalida dell'ordine.

Le interazioni di base dell'utente con l'applicazione sono descritte nella figura 8.

OrderMe si occupa anche della parte di gestione dell'amministrazione, nella quale l'utente con il ruolo di amministratore ha la possibilità di manipolare gli oggetti presenti nell'applicazione. Per manipolazione ci si riferisce alle quattro operazioni di base che possono essere svolte su un oggetto: visualizzare, eliminare, modificare e aggiungere. Tali operazioni

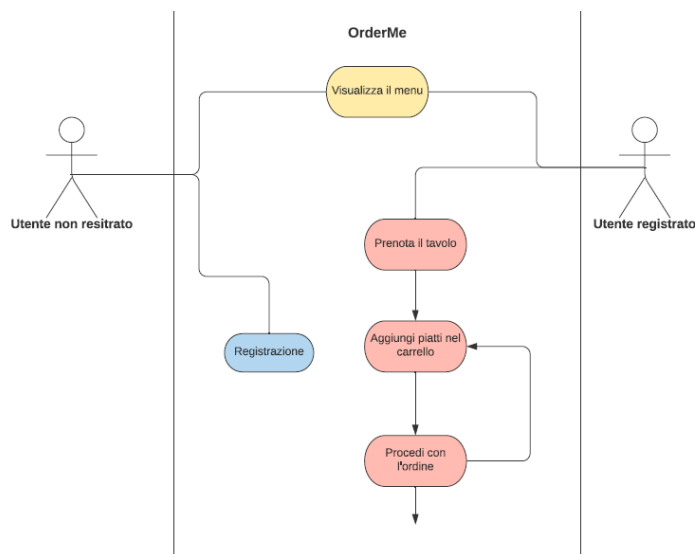


Figura 8: Interazioni di base dell'utente nell'applicazione

sono prodotte automaticamente dalle strutture di partenza del generatore di codice, il *Bake* di CakePHP, per poi essere ulteriormente raffinate dal sviluppatore in modo tale da implementare le funzionalità desiderate.

OrderMe è reperibile ai seguenti indirizzi web:

- <http://orderme.caiupi.com/> : Un esempio del funzionamento dell'applicazione.
- <https://github.com/caiupi/orderme> : I codici sorgente dell'applicazione.

3.2 IL MODELLO RELAZIONALE

CakePHP è un framework fortemente orientato agli oggetti (ORM), dove i dati del database devono essere rappresentati come oggetti. Tra dei suoi 'ingredienti' principale vi sono le convenzioni nel rappresentare le tabelle del database per esempio, ogni tabella deve essere provvista di una chiave primaria. Nel costruire il modello entità-relazione (ER) del database bisogna considerare questi due aspetti fondamentali e la progettazione del database dell'applicazione deve rispettarli. Di conseguenza la modellazione del database si differenzia, anche se di poco, da quello che poteva essere una progettazione classica del modello ER. In OrderMe si possono individuare cinque concetti/entità che formano lo scheletro del database:

- **users:** Rappresenta l'insieme degli utenti registrati, l'utente normale e l'utente che rappresenta amministratore dell'applicazione;
- **desks:** L'insieme dei tavoli del ristorante, dove una delle proprietà è la disponibilità;
- **dishes:** Questo insieme rappresenta il menu del ristorante composto dai diversi piatti;
- **carts:** Rappresenta il carrello di un utente registrato. Raccoglie tutti i piatti che l'utente desidera ordinare;
- **orders:** Collezione di piatti del menu che realmente sono ordinati dall'utente.

Un'altra convenzione è il fatto che le chiavi esterne (*foreign key*) sono riconosciute dal generatore di codice se terminano con *_id*.
Tenendo presente le considerazioni sopra elencate si ha il seguente schema ER come descritto nella figura 9

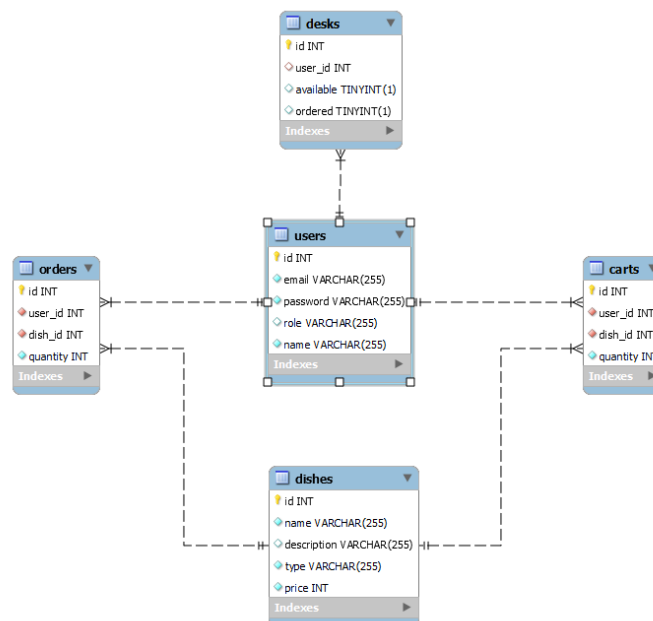


Figura 9: Schema ER del database di OrderMe

3.3 GENERAZIONE DEL CODICE TRAMITE LA CONSOLE

Una volta installato, CakePHP offre un gran numero di plugin (componente software) e prima di generare il codice si devono installare quelli che si utilizzano nell'applicazione.

3.3.1 Autenticazione

OrderMe utilizza il *pugin* : *Authentication*, reperibile tramite il *Composer* (Gestore delle dipendenze in PHP), ed una volta configurato gestisce le operazioni di registrazione, login e logout degli utenti registrati.

OrderMe prevede tre tipi differenti di utenti: quelli non registrati, i registrati e l'utente con il ruolo di amministrazione, dove gli utenti registrati sono distinguibili tramite l'attributo ruolo (*role*):

- `$user->role='user'`: Impostato al valore user per l'utente normale;
- `$user->role='admin'` : Per l'amministratore dell'applicazione.

3.3.2 Cake Bake OrderMe

Avendo tutti gli 'ingredienti' necessari si può procedere con la preparazione della torta (*cake bake*), utilizzando la console si genera lo scheletro dell'applicazione. Con il comando:

```
bin/cake bake all users
```

si generano i seguenti file PHP per users:

- `UserTable User` : Rappresentano i *Model*;
- `UsersController` : Il *Controller* che contiene le azioni;
- `CRUD templates` : Il *View*;
- `TestCases` per ogni classe generata e le *Fixture*.

Nello stesso modo si generano i file necessari anche per le altre quattro tabelle del database, rispettando i legami delle chiavi esterne (*foreign key*) fra le tabelle.

All'entità User si aggiunge la funzione seguente che offre protezione *hash* per le password in modo tale da non essere visibili a malintenzionati.

```
class User extends Entity
{
    //other methods
    protected function _setPassword(string $password)
    {
        $hasher = new DefaultPasswordHasher();
        return $hasher->hash($password);
    }
}
```

3.4 IMPLEMENTAZIONE DELL'APPLICAZIONE

Basandosi sullo scheletro precedentemente generato si sviluppa l'applicazione dove di seguito sono esposti i punti più interessanti.

3.4.1 Barra del Menu dell'applicazione

Nella figura 10 si ha una descrizione della barra del menu dell'applicazione. Troviamo qui le cinque entità del database:



Menu | Tables | My Cart | My Orders | Profile | Login | Logout

Figura 10: La barra di menu dell'applicazione OrderMe

- *Menu*: L'insieme dei piatti del ristorante;
- *Tables*: L'insieme dei tavoli;
- *My Cart*: Il carrello virtuale ovvero l'insieme dei piatti che l'utente seleziona;
- *My Orders*: Gli ordini effettuati dell'utente;
- *Profile*: Il profilo dell'utente che offre la possibilità di aggiornare i dati di registrazione.

Sono presenti, inoltre i *link*: *Login* e *Logout* che rappresentano le operazioni di base di autenticazione che operano su la tabella *users*.

Tramite la sottolineatura degli elementi della barra del menu ci si accorge su che entità si è posizionati.

3.4.2 Il menu del ristorante

Ogni utente che arriva nel sito, anche se non registrato, può consultare il menu e i suoi piatti. Di conseguenza per queste informazioni non è necessaria l'autenticazione. Nella classe `DishesController.php` si implementa il metodo `beforeFilter()` il quale indica che per la consultazione del menu (azione in `index`) e visualizzazione dei singoli piatti (azione in `view`) non è richiesta l'autenticazione.

```
public function beforeFilter(\Cake\Event\EventInterface $event)
{
    parent::beforeFilter($event);
    $this->Authentication->
        addUnauthenticatedActions(['index', 'view']);
}
```

L'amministratore è il solo ad avere diritto all'aggiunta o all'eliminazione dei piatti dal menu del ristorante.

3.4.3 Riservare un tavolo in OrderMe con MVC

Nella maggior parte delle azioni svolte dall'utente viene chiesto l'accesso, tramite il seguente codice posto all'inizio delle funzioni e successivamente si controlla lo stato di autenticazione.

```
if (!$this->Authentication->getResult()->isValid()) {
    $this->Flash->error(__('First you need to login.'));
}
```

Se l'utente non è autenticato sarà reindirizzato alla pagina di login, e contemporaneamente avvisato della richiesta evasa.

Una volta autenticato, l'utente può consultare la lista dei tavoli disponibili raggiungibile all'indirizzo `http:// ./.orderme/desks/` e la logica dell'azione si trova nel file `DesksController.php`, precisamente nel metodo `index()`. Inizialmente si verifica che l'utente non abbia prenotato altri

tavoli interrogando il *Model* rispettivo. Nel caso in cui abbia già prenotato un tavolo, sarà avvisato con i dati della prenotazione.

```
public function index()
{
    //other code
    $userDesk=$this->Desks->find()->select(['id'])
        ->where(['user_id'=>$user->id])->first();
    if(!is_null($userDesk)){
        $this->Flash->set(__($user->name.', you have already reserved
            table: '.$userDesk->id));
    }
    $desks = $this->paginate($this->Desks
        ->find()->where(['available'=>true]));
    $this->set(compact('desks'));
}
```

Successivamente viene creata la variabile `$desks`, che contiene i tavoli disponibili e che il *Controller* passerà al *View*.

In questo caso è `templates/Desks/index.php` il quale ha il compito di impaginarli all'utente.

```
<table>
  <thead>
    <tr>
      <th><?= $this->Paginator->sort('Table number') ?></th>
      <th class="actions"><?= __('Actions') ?></th>
    </tr>
  </thead>
  <tbody>
    <?php foreach ($desks as $desk): ?>
      <tr>
        <td><?= $this->Number->format($desk->id) ?></td>
        <td class="actions">
          <?= $this->Html->link(__('View'), ['action' => 'view',
              $desk->id]) ?>
        </td>
      </tr>
    <?php endforeach; ?>
  </tbody>
</table>
```

I link in OrderMe rispettano le convenzioni CakePHP, dove tramite la

classe HTML si creano marcatori (*markup*) ben formattati che utilizzano lo stile della pagina CSS già configurato.

Consultando i tavoli disponibili. l'utente ha la possibilità di effettuare una prenotazione. Tale azione è gestita dal metodo `reserve()` del *Controller* di desks che prende come argomento l'indice del tavolo che si desidera prenotare.

```
public function reserve($id = null)
{
    //other code
    $desk = $this->Desks->get($id, ['contain' => [],]);
    $desk->available = false;
    $desk->user_id=$this->request->getAttribute('identity')->id;
    if ($this->Desks->save($desk)) {
        $this->Flash->success(__('The desk number: '.$desk->id.' has been reserved.'));
        return $this->redirect(['controller'=>'Carts','action' => 'index']);
    }
    $this->Flash->error(__('The desk could not be reserved. Please, try again.'));
}
```

Crea l'oggetto contenuto nella variabile `$desk` che corrisponde al tavolo selezionato, assegnandolo all'utente e cambiando lo stato a non disponibile. Per l'operazione di memorizzazione dell'oggetto si richiama il metodo `save()` di CakePHP che gestisce anche eventuali eccezioni.

Una volta memorizzato l'oggetto con il nuovo stato, l'utente sarà reindirizzato nella pagina relativa alla gestione del carrello virtuale.

3.4.4 *Cart: Il carrello*

La logica del carrello virtuale è racchiusa nel *Controller* `CardsController`. Prima di ogni interazione con gli oggetti di questa classe viene eseguito il metodo `beforeFilter()`, il quale verifica se l'utente ha eseguito l'accesso e successivamente verifica la presenza di un tavolo prenotato dallo stesso, interrogando la tabella desks. In assenza della prenotazione, sarà reindirizzato ad eseguire tale azione.

```
public function beforeFilter(\Cake\Event\EventInterface $event)
{
    parent::beforeFilter($event);
```

```

if(!$this->Authentication->getResult()->isValid()){
    $this->Flash->error(__('First you need to login.'));
}else {
    $userId = $this->request->getAttribute('identity')->id;
    $userDesk = $this->Carts->Users->Desks->find()->select(['id'])
        ->where(['user_id' => $userId])->first();
    if (is_null($userDesk)) {
        $this->Flash->error(__('First you need to reserve a table.'));
        return $this->redirect(['controller' => 'desks', 'action' =>
            'index']);
    }
}
}
}

```

Nel *Controller* sono implementati i metodi `add()` e `delete()`, che sono responsabili dell'azione di aggiunta o rimozione di un piatto dal carrello. Di seguito si può osservare la logica dell'azione di aggiunta del piatto.

```

public function add($dishId): ?\Cake\Http\Response
{
    $cart = $this->Carts->newEmptyEntity();
    $cart->user_id=$this->request
        ->getAttribute('identity')->id;
    $cart->dish_id=$dishId;
    $cart->quantity=1;
    if ($this->Carts->save($cart)) {
        $this->Flash->success(__('The dish has been added to the cart.'));
        return $this->redirect(['action' => 'index']);
    }
    $this->Flash->error(__('The dish could not be added. Please, try
        again.'));
    return $this->redirect(['action' => 'index']);
}

```

Anche in questo caso si adoperano i metodi forniti dal framework, dove inizialmente si crea un oggetto vuoto per poi inizializzare i suoi attributi e infine si esegue il salvataggio tramite la funzione `save()` del CakePHP. L'utilizzo di tali funzioni è una scelta più conveniente della classica funzione di aggiunta del record direttamente nel database, poiché esse permettono di gestire meglio le eccezioni e gli errori generati. OrderMe avvisa l'utente con tre diversi tipi di messaggi che sono di colore:

- Blu: In caso di una semplice comunicazione o di un avviso;
- Verde: Quando l'azione svolta è andata a buon fine;
- Rosso: Se l'utente sta eseguendo un'azione non consentita oppure si è in presenza di un errore come ad esempio immettere un ordine quando il carrello è vuoto.

Tali messaggi sono possibili mediante l'utilizzo di javascript implementato tramite la classe Flash di CakePHP come descritto nella figura 11.

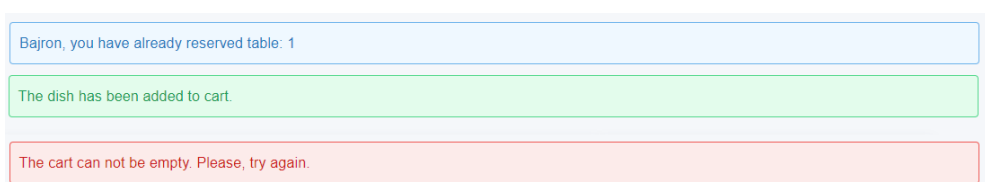


Figura 11: Le notifiche flash

3.4.5 Gli ordini

Quando l'utente esegue l'ordine, l'applicazione richiama il metodo `press()` del *Controller*: `OrdersController.php`. La funzione ottiene dal plugin dell'autenticazione, `id`, interroga il database per vedere se il carrello è vuoto reindirizzando l'utente ad aggiungere dei piatti. In caso contrario, sposta i record dal carrello alla tabella degli ordini.

```
public function press()
{
    $userId=$this->request->getAttribute('identity')->id;
    $cartsUser=$this->Orders->Users->Carts->find()
        ->where(['user_id'=>$userId]);
    if($cartsUser->count()==0){
        $this->Flash->error(__('The cart can not be empty. Please, try
            again.'));
        return $this->redirect(['controller'=>'Carts','action' => 'index']);
    }
    foreach ($cartsUser as $row) {
        $order = $this->Orders->newEmptyEntity();
        $order->user_id=$userId;
        $order->dish_id=$row->dish_id;
```

```

$order->quantity=$row->quantity;
if ($this->Orders->save($order)) {
}else{
    $this->Flash->error(__('The order could not be saved. Please, try
        again.'));
    }
}
foreach ($cartsUser as $row){
    if ($this->Orders->Users->Carts->delete($row)){
    }else{
        $this->Flash->error(__('The cart elements can not be eliminate.
            Please, try again.'));
        }
    }
}
$this->Flash->success(__('The order has been created.'));
$orders = $this->paginate($this->Orders);
$this->set(compact('orders'));
return $this->redirect(['action' => 'index']);
}

```

Successivamente svuota il carrello ed infine reindirizza l'esecuzione nella funzione, che in seguito permette di visualizzare i record.

Quando il cliente ha concluso il pasto, l'applicazione simula il pagamento considerando solo le operazioni necessarie che coinvolgono OrderMe. La funzione `payment()` si occupa di tale operazione.

```

public function payment()
{
    $userId=$this->request->getAttribute('identity')->id;
    $desksUser=$this->Orders->Users->Desks->find()
        ->where(['user_id'=>$userId])->first();
    if($desksUser==null){
        $this->Flash->error(__('You dont have order
            any dishes. First you have to order.'));
        return $this->redirect(['controller'=>'Carts','action' => 'index']);
    }
    //Empty the cart and the order records of the user
    $ordersUser=$this->Orders->find()->where(['user_id'=>$userId]);
    $cardsUser=$this->Orders->Users->Carts
        ->find()->where(['user_id'=>$userId]);
    foreach ($ordersUser as $row){
        if ($this->Orders->delete($row)){
        }else{
        }
    }
}

```

```

        $this->Flash->error(__('The orders could
            not be deleted. Please, try again.'));
    }
}
foreach ($cardsUser as $row){
    if ($this->Orders->Users->Carts->delete($row)){
    }else{
        $this->Flash->error(__('The orders could
            not be deleted. Please, try again.'));
    }
}
$desksUser->ordered=true;
$desksUser->available=true;
$desksUser->user_id=null;
$this->Orders->Users->Desks->save($desksUser);
$this->Flash->success(__('Thank you. See you soon.'));
return $this->redirect(['controller'=>'Dishes','action' => 'index']);
}

```

Elimina gli ordini dell'utente, in seguito svuota il carrello associato. Infine, libera il tavolo rendendolo disponibile ed in questo modo si elimina l'associazione con l'utente.

3.4.6 Amministratore e SQL injection

Progettare e implementare la parte che si occupa dell'amministrazione dell'applicazione è importante ma si è quindi preferito non aggiungere una nuova barra di menu sfruttando quella esistente.

Le funzioni `index()` dei *Controller* iniziano con la sequenza di codice, la quale verifica che l'utente sia effettivamente un amministratore. In caso positivo si chiama la funzione `admin()` dello stesso *Controller*.

```

public function index()
{
    $user= $this->request->getAttribute('identity')->getOriginalData();
    if ($user->isAdmin()){
        $this->Flash->set(__('You are redirected to the administration Orders
            page'));
        return $this->redirect(['action' => 'admin']);
    }
    //other code

```

```
}

```

La funzione `admin()` ha lo stesso ruolo di `index()`, con la differenza che in questo caso, l'amministratore ha tutti gli strumenti per accedere e manipolare gli oggetti a quel particolare *Controller*.

Un esempio della funzione per la tabella `desks` è il seguente:

```
public function admin(){
    $user= $this->request->getAttribute('identity')->getOriginalData();
    if (!$user->isAdmin()){
        $this->Flash->error(__('You need to be an administrator for access
        the page.'));
        return $this->redirect(['action' => 'index']);
    }
    $this->paginate = ['contain' => ['Users'], ];
    $desks = $this->paginate($this->Desks);
    $this->set(compact('desks'));
}
```

Tale azione trasmette al *View* tutti gli elementi di quella tabella, come in questo caso i dati nella variabile `$desks` e successivamente sarà visualizzato tramite il file `templates/Desks/admin`. L'utente normale è abilitato alla sola visualizzazione dell'elenco dei tavoli disponibili, mentre l'amministratore vede tutto il contenuto della tabella. Per ogni record, inoltre, si hanno i link per effettuare le quattro azioni predefinite.

Gli attacchi di tipo SQL injection, in PHP, sono un problema serio. Un utente malizioso che conosce la struttura di CakePHP può accedere e modificare gli oggetti dell'applicazione. Per ovviare a tale problema è necessario tentare di rispettare le seguenti regole:

- I dati passati al *View* tramite il metodo `set()` per poi essere visualizzati all'utente, devono contenere l'informazione necessaria senza includere informazioni sensibili.
- Dato che l'utente interagisce con l'applicazione tramite l'indirizzo URL, tutte le azioni pubbliche devono controllare il ruolo dell'utente che accede e offrire il giusto servizio. Per esempio un utente normale tenta di accedere all'indirizzo `http://.../orderme/desks/add`, che corrisponde all'azione dell'amministratore di aggiungere un tavolo, sarà avvisato e reindirizzato ad un'altra pagina.

3.5 TESTING E CODE COVERAGE

Il *testing* del software è un procedimento importante e indispensabile per individuare le carenze di correttezza, completezza e affidabilità. Per testare le componenti di OrderMe si utilizza il metodo di *Unit tests* appoggiandosi sulle funzioni di CakePHP che a loro volta si basano sul framework PHPUnit.

Il processo di *testing*, in questo elaborato, ha come obiettivo di offrire buone garanzie sul corretto funzionamento del codice. Si cercherà di analizzare se le funzioni che si andrà a testare, eseguono le azioni per cui sono costruite.

I punti da studiare sono le componenti logiche, pertanto il fase di *testing* si concentrerà sui *Controller* dell'applicazione. Per ognuno di essi si costruisce una classe di test che estende *TestCase*. Il test utilizza le asserzioni che sono delle affermazioni relative ad una condizione che deve essere vera.

Durante tale processo si farà uso delle seguenti funzioni di CakePHP per simulare le richieste HTTP:

- `get()` : Inoltra una richiesta GET;
- `delete()` : Trasmette una richiesta di DELETE;
- `patch()` : Invia una richiesta di PATCH.

I *files* prodotti sono contenuti nella cartella *test*.

3.5.1 Configurazione per il testing

Tenendo presente che si affronta un numero considerevole di test è importante configurarli nel modo adeguato. Per questo motivo le classi di *TestCase* implementano la funzione di *setUp()*. Essa viene eseguito prima di ogni test ed offre gli strumenti necessari.

```
public function setUp(): void
{
    parent::setUp();
    $this->loadRoutes();
    $this->Dishes = $this->getTableLocator()->get('Dishes');
    $this->Users = $this->getTableLocator()->get('Users');
    $this->Desks = $this->getTableLocator()->get('Desks');
```



```

$this->configRequest(['environment' => ['HTTPS' => 'on']]);
$this->enableRetainFlashMessages();
$this->enableCsrfToken();
$this->enableSecurityToken();
}

```

La funzione crea i collegamenti con il database e attiva, inoltre, delle opzioni di CakePHP che altrimenti genererebbero errori durante la fase di *testing*.

Per simulare l'autenticazione dell'utente nell'applicazione, si adopera la funzione `login()` che prende come input l'id dell'utente memorizzato sui record delle *fixtures*.

```

protected function login($userId = 2)
{
    $user = $this->Users->get($userId);
    $this->session(['Auth' => $user]);
}

```

Le *fixtures* della tabella `users` sono creati in modo tale da cercare di coprire tutte le possibili tipologie d'utenti. I record corrispondono logicamente alle seguenti categorie:

- 'id'=> 1 : Utente registrato che non ha prenotato un tavolo;
- 'id'=> 2 : Utente registrato con il ruolo di amministratore;
- 'id'=> 3 : Utente registrato che ha prenotato un tavolo .

3.5.2 Esempi di testing sul menu del ristorante

Il test eseguito ha la finalità di verificare l'invio di una richiesta HTTP, dall'utente non registrato all'applicazione, per consultare il menu del ristorante.

```

class DishesControllerTest extends TestCase
{
    \\other functions here
    public function testIndex(): void
    {
        $this->get('/dishes/index');
        $this->assertResponseOk();
    }
}

```

```

        $this->assertResponseContains('Dishes');
        $this->assertNoRedirect(); //No redirection
        $this->assertSession('Login if you need to order.',
            'Flash.flash.o.message');
    }
}

```

Infatti dall'ultima asserzione si è affermato che l'applicazione invita l'utente ad effettuare il login.

Contrariamente, se l'amministratore (id=2) richiede d'intraprendere la stessa azione,

```

public function testIndexWithLoginAdmin()
{
    $this->login(2);
    $this->get('/dishes/index');
    $this->assertRedirect(['controller' => 'Dishes', 'action' => 'admin']);
}

```

sarà reindirizzato alla funzione admin().

Se un utente normale registrato richiede una funzione riservata all'amministratore verrà reindirizzato alla pagina del menu iniziale.

```

public function testAdminWithLoginNormalUser()
{
    $this->login(1);
    $this->get('/dishes/admin'); //access admin function
    $this->assertSession('You need to be an administrator for access the
        page.', 'Flash.flash.o.message');
    $this->assertRedirect(['controller' => 'Dishes', 'action' => 'index']);
}

```

Dal test si osserva che le ultime due asserzioni affermano la comunicazione dell'applicazione tramite i messaggi Flash e il relativo reindirizzamento.

3.5.3 *Testing sulla prenotazione del tavolo*

Se l'utente che ha già prenotato un tavolo e richiede di prenotarne un altro, questa azione non è permessa.

```

class DesksControllerTest extends TestCase

```

```

{
    public function testReserveWithUserThatHaveReservedATable()
    {
        $this->login(3);
        $this->get('/desks/reserve/3');
        $this->assertSession('The desk could not be reserved. You have
            already reserved a table.', 'Flash.flash.o.message');
        $this->assertRedirect(['controller' => 'Desks', 'action' => 'index']);
    }
    \\other function here
}

```

Come si osserva, le asserzioni affermano l'avviso dell'impossibilità di effettuare la richiesta.

Differente è il comportamento in caso della prenotazione del tavolo dall'utente che non ha riservato dei tavoli.

```

public function testReserve()
{
    $this->login(1);
    $this->get('/desks/reserve/3');
    $this->assertSession('The desk number: 1 has been reserved.',
        'Flash.flash.o.message');
    $this->assertRedirect(['controller' => 'Carts', 'action' => 'index']);
}

```

Le asserzioni confermano la prenotazione tramite la notifica della classe Flash e indirizza l'utente verso il carrello per proseguire nell'effettuare l'ordine.

3.5.4 *Testing delle operazioni sul carrello*

Il seguente test controlla che, in assenza di una prenotazione del tavolo, se un utente registrato richiede di aggiungere dei piatti al carrello allora l'applicazione deve accorgersi di quest'azione. .

```

class CartsControllerTest extends TestCase
{
    public function testAddWithLoginNormalUserWithoutTable(): void
    {
        $this->login(1);
        $this->get('/carts/add/');
    }
}

```

```

        $this->assertResponseSuccess();
        $this->assertRedirect(['controller' => 'Desks', 'action' => 'index']);
        $this->assertSession('First you need to reserve a table.',
            'Flash.flash.o.message');
    }
    \\other functions
}

```

Come si nota dalle asserzioni, l'utente è avvisato dell'impossibilità dell'azione e successivamente reindirizzato ad effettuare la prenotazione. Nel caso seguente, l'utente effettua una richiesta di `post()` per aggiungere un piatto al suo carrello.

```

public function testAdd(): void
{
    $this->login(3);
    $data = [
        'id' => 4,
        'user_id' => 3,
        'dish_id' => 2,
        'quantity' => 1,
    ],]; // the new record to add to the database
    $this->post('/carts/add/2', $data[0]);
    $this->assertResponseSuccess();
    $query = $this->Carts->find()->where(['id' => 4]);
    $result = $query->enableHydration(false)->toArray();
    $this->assertEquals($result, $data);
}

```

Le asserzioni affermano il risultato di quest'azione e in seguito controlla che l'oggetto effettivamente è stato aggiunto nel database.

In seguito, si esegue il test con l'utente che ha prenotato un tavolo e desidera eliminare un oggetto dal proprio carrello.

```

public function testDeleteWithLoginNormalUserWithTable(): void
{
    $this->login(3);
    $this->delete('/carts/delete/3'); //delete the third element
    $this->assertResponseSuccess();
    $this->assertSession('The dish has been deleted.',
        'Flash.flash.o.message');
    $query = $this->Carts->find();
    $result = $query->enableHydration(false)->toArray();
}

```

```

$result_count = count($result);
$expected_count = 2;
$this->assertEquals($expected_count, $result_count);
}

```

In questo caso si ha una richiesta di eliminare un record, `delete()`, e le affermazioni confermano l'avvenuta eliminazione avvisando l'utente. Il test, inoltre verifica che dal database è stato rimosso un elemento.

3.5.5 Testing l'azione di invio dell'ordine

L'invio dell'ordine è gestito dalla classe `OrdersControllerTest`. Il test controlla che all'utente sia inviata la pagina html giusta.

```

class OrdersControllerTest extends TestCase
{
    public function testPressWithLoginUserWithATable()
    {
        $this->login(3);
        $this->get('/orders/press'); // make the order
        $this->assertHeaderContains('Content-Type', 'html');
        $this->assertRedirectContains('/orders');
        $this->assertRedirect(['controller' => 'Orders', 'action' =>
            'index']);
        $query = $this->Orders->find();
        $result = $query->enableHydration(false)->toArray();
        $result_count = count($result);
        $expected_count = 4;
        $this->assertEquals($expected_count, $result_count);
    }
    //other function
}

```

L'ultima asserzione afferma che nella tabella degli ordini è stato aggiunto un nuovo oggetto.

Una volta effettuato l'ordine, l'utente non ha la possibilità di eliminare i record.

```

public function testDeleteWithLoginNormalUserWithTable(): void
{
    $this->login(3);
    $this->delete('/orders/delete/3');
}

```

```
$this->assertSession('You need to be an administrator for access the  
page.', 'Flash.flash.o.message');  
$this->assertRedirect(['controller' => 'Orders', 'action' => 'index']);  
}
```

Si osserva che se tenta di accedere alla funzione che si occupa della rimozione dei piatti dall'ordine, l'applicazione lo avvisa negando l'accesso e reindirizzandolo in un'altra pagina.

Nella seguente funzione, inizialmente un utente che non ha un tavolo cerca di ordinare e il test conferma che l'applicazione gestisce la richiesta.

```
public function testPressWithLoginUserWithoutTable(): void  
{  
    $this->login(1);  
    $this->get('/orders/press'); //press an order without table  
    $this->assertResponseSuccess();  
    $this->assertRedirect(['controller' => 'Desks', 'action' => 'index']);  
    $this->assertSession('First you need to reserve a table.',  
        'Flash.flash.o.message');  
    $this->get('/desks/reserve/1'); //reserve an available table  
    $this->assertResponseSuccess();  
    $this->assertSession('The desk number: 1 has been reserved.',  
        'Flash.flash.o.message');  
    $this->assertRedirect(['controller' => 'Carts', 'action' => 'index']);  
    $this->get('/orders/press'); //press an order with empty cart  
    $this->assertSession('The cart can not be empty. Please, try again.',  
        'Flash.flash.o.message');  
}
```

Successivamente, il test conferma positivamente la richiesta dell'utente di effettuare una prenotazione. Quando l'utente cerca di eseguire un ordine con il carrello vuoto, le asserzioni confermano la giusta risposta dell'applicazione.

Come si evince, *Unit tests* effettua il test sulle singole componenti, tuttavia quando si parla di *testing* su operazioni complesse rappresentate da più funzioni come quella precedente, allora è necessario l'utilizzo di altri approcci, come *Integration Test*.

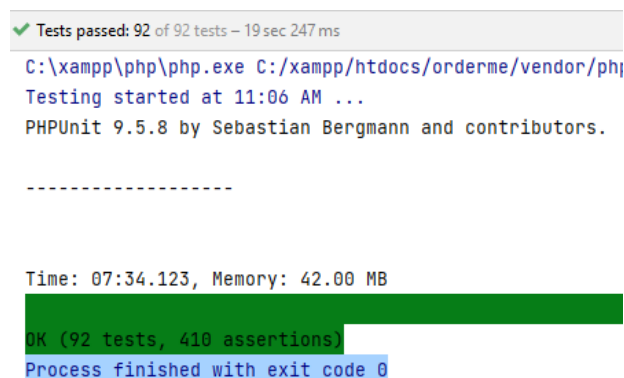
3.5.6 Code coverage

Completata la fase di implementazione dei test, con la seguente linea di comando si eseguono tutti i tests di OrderMe:

```
.\vendor\bin\phpunit tests
```

Il risultato si osserva nella seguente figura 12.

Nel processo di *testing* si è cercato di analizzare molti aspetti dell'applicazione



```
✓ Tests passed: 92 of 92 tests – 19 sec 247 ms
C:\xampp\php\php.exe C:/xampp/htdocs/orderme/vendor/phpunit/phpunit
Testing started at 11:06 AM ...
PHPUnit 9.5.8 by Sebastian Bergmann and contributors.

-----

Time: 07:34.123, Memory: 42.00 MB
OK (92 tests, 410 assertions)
Process finished with exit code 0
```

Figura 12: L'insieme dei test effettuati su OrderMe

cazione in modo tale da ricoprire una buona parte delle sue funzionalità. Una particolare attenzione è stata dedicata a una parte della sicurezza (*security*), dove si è verificato che gli utenti interagiscono in modo corretto. Gli accessi alle funzioni sono suddivisi secondo i ruoli, cercando di limitare l'accesso in determinate operazioni all'utente normale senza tuttavia restringere il ruolo dell'amministratore. Utilizzando i test si è verificato che gli utenti non in possesso di alcune caratteristiche non possono accedere a determinate funzioni.

Tramite gli avvisi dell'applicazione, si è tentato di verificare la notifica corretta da parte di OrderMe all'utente, e successivamente il giusto orientamento per eseguire le azioni in una sequenza collaudata.

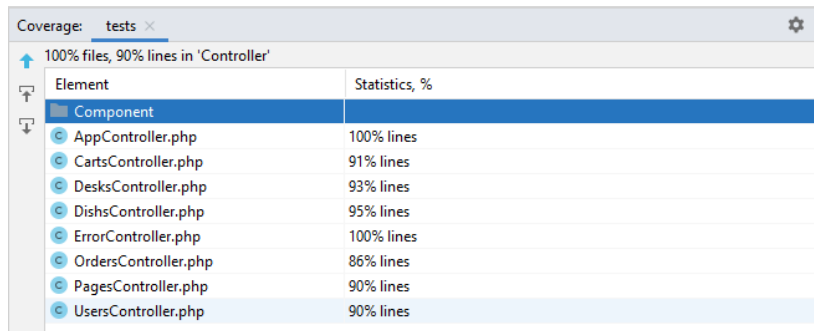
Tramite il *testing* sono state analizzate le operazioni di manipolazione dei dati, come l'eliminazione o l'aggiunta dei nuovi record.

Il processo di *testing* si sviluppa in parallelo con la copertura del codice (*code coverage*). Quest'ultimo fornisce delle indicazioni sulla quantità di *UnitTest* eseguite sull'applicazione, la percentuale della linea di codice coperta dall'esecuzione dei test conclusi positivamente. Lo scopo dunque

è aumentare al massimo la *coverage*, testando più componenti dell'applicazione.

Nella seguente figura 13 si ha la percentuale di *code coverage* per i *Controller* di OrderMe.

La fase di *testing* non è terminata, è opportuno verificare ulteriori aspetti,



Coverage: tests	
100% files, 90% lines in 'Controller'	
Element	Statistics, %
Component	
AppController.php	100% lines
CartsController.php	91% lines
DesksController.php	93% lines
DishesController.php	95% lines
ErrorController.php	100% lines
OrdersController.php	86% lines
PagesController.php	90% lines
UsersController.php	90% lines

Figura 13: Risultato della copertura dei *Controller* di OrderMe

per esempio testare l'autorizzazione dei dati, eseguire i test su gli *View* oppure la validazione dei dati immessi dall'utente. Anche qualora si raggiungesse l'ipotetica cifra di 100% di copertura del codice di una componente software, questo non è una garanzia di assenza *bug*.

La *code coverage*, probabilmente, è la metrica più importante che si può estrarre dall'esecuzione degli *UnitTest*. Indica la quantità dei test eseguiti, ma non è necessariamente una buona metrica che determina la buona implementazione dei test.

Considerando la percentuale di *code coverage* raggiunta sulle componenti testate basate sulle funzioni analizzate, è possibile affermare che è stata raggiunta una buona copertura.

CONCLUSIONI

La trattazione presentava l'obiettivo di sviluppare un'applicazione web con il framework CakePHP. L'intento era quello di capire se sia effettivamente possibile realizzare software corretto nelle sue funzionalità utilizzando il pattern architetturale Model View Controller (MVC) implementando il paradigma di programmazione (OOP) utilizzando CakePHP. Per questo è stato realizzato un prototipo di applicazione, OrderMe, che simula un applicativo web per la ristorazione implementando i concetti del framework. Nella fase di *testing* si è verificato la correttezza delle sue componenti nelle sue funzionalità con particolare attenzione alla sicurezza. Tramite il *code coverage* si è mostrato che si può raggiungere un buon livello di copertura del codice.

Dall'indagine risulta dunque che CakePHP è un framework che offre al programmatore tutti gli strumenti adatti per sviluppare applicazioni web corretti, ossia, si comporta secondo quanto previsto nelle sue funzionalità. Durante la fase di realizzazione si è riscontrato il problema di una documentazione non adeguata quando si tratta di implementare tecniche di *testing* in CakePHP. Il passaggio recente nella nuova versione ha invalidato la documentazione precedente e, in letteratura, si sente l'assenza di pubblicazioni che approfondiscono tali tematiche. Diversamente, la sua piccola ma attiva comunità è notevolmente utile per riempire questo vuoto.

BIBLIOGRAFIA

- [1] Berners-Lee Tim - *World-wide web* - Computer Networks and ISDN Systems, Volume 25, 1992, Pages 454-459
- [2] Nixon Robin *Learning PHP, MySQL and JavaScript* - O'reilly, 4th edition, 2014
- [3] Lockhart Josh - *Modern PHP New Features and Good Practices* - O'reilly, 2015
- [4] Evans Cal - *Guide to Programming with Zend Framework* - Marco Tabini & Associates, Inc., 2008
- [5] Porebski, Przystalski, Nowak - *Building PHP Applications with Symfony, CakePHP, and Zend Framework* - Wiley, 2011
- [6] <https://reteinformaticalavoro.it/blog>
- [7] Delisle Marc - *Mastering phpMyAdmin 2.11 for Effective MySQL Management* - Packt Publishing, 2008
- [8] DuBois Paul - *MySQL Cookbook* - O'reilly, Third Edition, 2014
- [9] <https://httpd.apache.org/>
- [10] Cake Software Foundation - *CakePHP Cookbook Documentation* - Cake Software Foundation, 2021
- [11] <https://cakephp.org/>
- [12] Martin James - *Managing the data-base environment* - Prentice-Hall, 1983
- [13] Chan K., Omokore J., Millar R.K., - *Practical CakePHP Projects* - Apress, 2009
- [14] Laurent Andrew M. St. - *Understanding Open Source and Free Software Licensing* - O'reilly ,2004
- [15] Gamma E, Vlissides J. - *Design Patterns: Elements of Reusable Object-Oriented Software* - Addison-Wesley, 1994

- [16] Kevin Dunglas - *Persistence in PHP with Doctrine ORM* - Packt Publishing, 2013
- [17] <https://phpunit.de/>
- [18] Bettini Lorenzo - *Test-Driven Development, Build Automation, Continuous Integration* - Learnpub, 2020
- [19] Bergmann Sebastian - *PHPUnit Manual* - 2020
- [20] Dasa Radharadhya - *Learn CakePHP With Unit Testing* - Apress, 2016
- [21] <https://github.com/cakephp/cakephp/tree/master/tests/TestCase>
- [22] <https://getcomposer.org/>