Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea in Scienze di Internet

PROGETTAZIONE E SVILUPPO DI UNA APPLICAZIONE WEB PER LA GESTIONE DELLE PRENOTAZIONI IN AMBITO RISTORATIVO

Tesi di Laurea in Basi di Dati

Relatore:
Chiar.mo Prof.
Danilo Montesi

Presentata da:
Nicola Buongiorno

Sessione III
Anno Accademico 2012-2013

Indice

Capitolo 1

Introduzione

Negli ultimi anni si è assistito ad uno sviluppo delle *ICT* (*Information and Comunication Technology*) e delle relative applicazioni come fulcro funzionale allo sviluppo; nonstante ciò gran parte dei consumatori e dei gestori di attività ristorative (ristoranti, pizzerie, trattorie ecc.) considerano l'impiego della tecnologia e internet una componente alternativa anzichè essenziale.

A partire dalle prime innovazioni tecnologiche si denota come l'impatto di internet abbia suscitato notevoli migliorie sia nel campo della gestione/progettazione che nel campo economico. Analizzando l'attuale periodo e confrontandomi con l'ambiente circostante ho potuto notare che circa l'80% delle strutture ristorative si sia dovuta adeguare ai tempi e sviluppare forme di competizione inerenti all'ambito che la tecnologia sta sviluppando.

L'obiettivo di questa tesi è la progettazione e lo sviluppo di una *web application* attinente la gestione delle prenotazioni dei tavoli per un'attività ristorativa, in un mercato del tipo *Business to Business (B2B)* ovvero le transazioni che si vengono a instaurare tra fornitore di servizio (*Web app*) e consumatore (attività ristorativa). *MyRBM (My Restaurant Booking Manager)* è il nome che ho scelto per la *Web app* dopo un'attenta ricerca di mercato che si è focalizzata oltre che sull'analisi del nome dei possibili *competitor*, anche sull'impatto che lo stesso può avere sui possibili utilizzatori.

1.1 My Restaurant Booking Manager (MyRBM)

Attraverso una semplice interfaccia web il gestore dell'attività ristorativa ha la possibilità di visualizzare, inserire e modificare prenotazioni dei tavoli per i propri clienti. Nel caso pertinente alla ristorazione, in accordo con i tempi che seguono e sempre una maggiore autonomia del consumatore che attraverso la mediazione di internet è sempre più informato e critico nei confronti della gestione, vuole spendere il minor tempo possibile per lo screening delle proposte di qualità/prezzo. Il "nuovo" gestore nell'epoca del post-moderno si trova a dover configurare un rapporto con il proprio cliente, sia il più immediato possibile, sia nel modo più esaustivo ed efficace possibile, abbattendo i costi di ricerca e di prenotazione (di un tavolo). Nella mia configurazione di *web application* ho ritenuto la semplicità e la trasparenza come assunto fondamentale di base, essendo questa la ricetta di un buon management e di soddisfazione del top management (ristoratore).

Partendo dal presupposto che il consumatore non vuole solo la facilità di utilizzo ma anche immediato riscontro pratico e la premiazione della fedeltà, mi sono servito di un database come raccolta dati e ho fatto sì che il gestore, attraverso l'utilizzo di un'interfaccia elettronica (*web*), abbia accesso immediato e uno storico delle prenotazioni del cliente per capire quando lo stesso può aver avuto un problema e magari offrirgli una particolare promozione in futuro. L'idea, seppur non troppo rivoluzionaria, vuole ampliare il raggio di azione del termine stesso *booking* che fin ora è stato quasi esclusivamente inteso come prenotazione di voli, hotel, appartamenti ecc. Ho voluto concentrare la mia ricerca verso un campo in fase di sviluppo, "BOOKING MANAGEMENT RESTAURANT"; ampliando il contesto semantico del termine proprio (*booking*/prenotazione), sfruttando le tecnologie attuali che vanno oltre il mero utilizzo fisico di carta e penna che in questi anni è stato causa di consolidati errori, oltre che grammaticali, di distrazione e negligenza da parte di un manager di un'attività ristorativa. Voglio mettere in evidenza che la gestione (es. ristorante) comprende, oltre gli aspetti economici di massimizzazione del profitto e quindi soddisfazione del cliente, anche l'organizzazione della sala in base alle esigenze di circostanza. In un esempio che vede la configurazione di un'attività ristorativa che utilizza *MyRBM* e che attraverso questa conosce e migliora i propri rapporti con la clientela, oltre che ottimizzare l'allocazione attra-

verso note aggiuntive, riesce a specificare e far sì che le prenotazioni rispecchino le diverse esigenze dei clienti. Si denota come il processo di *customer care* si venga così a consolidare vedendo i desideri del consumatore trasformati in una realtà ben delineata e precisa descritta da egli stesso nelle note; si pensi ai posti di prestigio e alla lotta per aggiudicarseli. Attraverso l'immagazzinamento delle conoscenze e preferenze e della frequenza di prenotazione, l'assegnazione di questi contesi "posti di onore" diviene non una questione prettamente economica ma di reciproca stima e fedeltà nei confronti del cliente. Voglio ulteriormente esplicitare la mia decisione di lasciare il libero arbitrio della gestione e della composizione della sala esclusivamente al *direttore di sala*, che di volta in volta deve affrontare nuove richieste di composizione e agglomeramento tavoli in modo da garantire la totale *customer satisfaction* [PreRog07], avviando così un processo di fidelizzazione che vede nella fedeltà stessa un reale guadagno (profitto) da parte del gestore dell'attività ristorativa, sapendo che un cliente pienamente fedele (soddisfatto) porta maggiore profitto piuttosto che andare a cercarne uno nuovo; la stessa fedeltà che intendo creare attraverso un flusso diretto con il manager dell'attività ristorativa, che riconoscendo semplicità e trasparenza nel mio servizio, si troverà a gestire il proprio locale in maniera ottimale abbattendo i costi di gestione e di tempo.

La mia web application si inserisce in un mercato (*restaurant booking*) altamente competitivo, che va dal programma/software progettato e sviluppato esclusivamente ad hoc e gestito dal personale con un dispositivo elettronico (es. *Direca*[1]), al sito che propone un elenco di ristoranti/pizzerie circoscritto in base alla località e alle caratteristiche (es. *book a restaurant*[2]). Studiando ed analizzando il mercato in questione e volendomi esclusivamente dedicare alla gestione del booking della ristorazione da parte del manager, ho sviluppato un sistema *erga omnes* utilizzabile gratuitamente da pc, smartphone e tablet e adattabile ad ogni tipo di attività ristorativa. Dalla mia ricerca risulta che all'estero ma soprattutto in Italia, non esistono applicazioni che offrono esclusivamente questo servizio e che siano economicamente accessibili, essendo la maggior parte inserite nelle due grandi categorie precedentemente descritte. Tutte queste applicazioni hanno in comune una serie di componenti aggiuntive che danno sì la possibilità di personalizzazione ma aggiun-

[1]Software di gestione ristoranti; URL: *http://www.direca.it/*
[2]Servizio online di prenotazione per vari ristoranti; *URL: http://www.bookarestaurant.com/*

gono elementi che possono distrarre il consumatore ed elementi di compilazione dati che sommati lo portano a un reale dispendio di tempo. Avendo semplificato e snellito i servizi sopra citati dalle funzionalità eccedenti e superflue, ho focalizzato la mia attenzione sugli aspetti cruciali che sono la semplicità di comprensione e di utilizzo. Il sito che più si avvicina a *MyRBM* è *simple Electronic Reservation Book*[3] *(simpleERB).*

La seguente tabella mostra come le volute mancanze nella mia applicazione siano dettate dalla ricerca costante di massimizzazione del tempo[DinFab05].

	prenotazione tavolo	cliente nuovo	gestione tavoli	gestione ordini	tempo di prenota-zione	facilità utiliz-zo	dispositivo
MyRBM	si	si	no	no	2	si	web
simpleERB	si	no	si	si	3	no	web
Direca	si	si	si	si	4	no	palmare ad hoc

Tempo di prenotazione = $\sum si$; si = 1; no = 0.

1.2 Applicazione web

Una applicazione web è una applicazione *client/server* per un ambiente *stateless*, cioè senza memoria, che utilizza le tecnologie internet. Con il termine *Web app* si descrive un'applicazione accessibile via web per mezzo di una network, come ad esempio una Intranet o attraverso la rete Internet. Pertanto, saper programmare per il web significa conoscere i diversi meccanismi e strumenti per conservare o passare i dati, detti parametri, tra le diverse pagine dell'applicazione web. In pratica una *Web-application*, è un programma che non necessita di essere installato nel computer in quanto esso si rende disponibile su un server in rete e può essere fatto funzionare attraverso un normale *Web browser* (es. Google Chrome, Mozilla Firefox, Opera, ecc.) in posizione di client. Il client, dopo aver instaurato una connessione con il server, invia la richiesta per un servizio; il server dopo aver

[3] *URL* del servizio *simpleERB: http://www.simpleerb.com*

elaborato i dati necessari rende disponibile al client il servizio richiesto. A differenza dei siti web statici (*HTML*), l'applicazione web viene realizzata con una o più tecnologie (*PHP, Ajax, Servlet, Database* ecc.) che permettono la creazione di un sito dinamico, cioè di un sito nel quale il contenuto delle pagine varia durante l'interazione.

Le applicazioni Web si pongono come valida alternativa alle tradizionali applicazioni Client/Server per vari motivi:

- facilità di distribuzione e aggiornamento: un'applicazione Web risiede interamente sul server, per cui la sua pubblicazione coincide con la distribuzione e l'aggiornamento ed è automaticamente disponibile a tutti gli utenti;
- accesso multipiattaforma: l'accesso all'applicazione è indipendente dall'hardware e dal sistema operativo utilizzato dagli utenti;
- riduzione del costo di gestione: l'uso di Internet come infrastruttura per un'applicazione Web riduce notevolmente sia i costi di connettività che i costi di gestione dei client;
- scalabilità: un'applicazione Web ben progettata può crescere insieme alle esigenze dell'azienda senza particolari problemi.

1.3 Riassunto dei prossimi capitoli

Nel secondo capitolo viene definito il documento di analisi dei requisiti composto dal glossario dei termini utilizzati, l'idea da sviluppare, i requisiti funzionali e non funzionali, i casi d'uso e i diagrammi di attività.

Il terzo capitolo descrive la tecnologia e gli strumenti che sono stati acquisiti ed impiegati per la realizzazione dell'applicazione : XAMPP, Apache, MySQL, PHP, HTML, SQL, JavaScript su piattaforma Microsoft; e descrive la progettazione della base di dati utilizzata.

Nel quarto capitolo viene descritta l'implementazione del sito con la spiegazione di ogni pagina prodotta e i test effettuati su di esso.

L'ultimo capitolo comprende le conclusioni del lavoro svolto, una stima del costo per produrre la Web app e la sezione degli sviluppi futuri.

Capitolo 2

Analisi dei requisiti

Un requisito è una scrittura più o meno formale e rigorosa di una caratteristica del sistema, fatta dallo specialista. La gestione dei requisiti (acquisizione, analisi, negoziazione, specifica, validazione) è il primo passo del processo di sviluppo e una delle fasi più critiche dello sviluppo software, perché influenza in modo diretto il successo o il fallimento dei progetti.

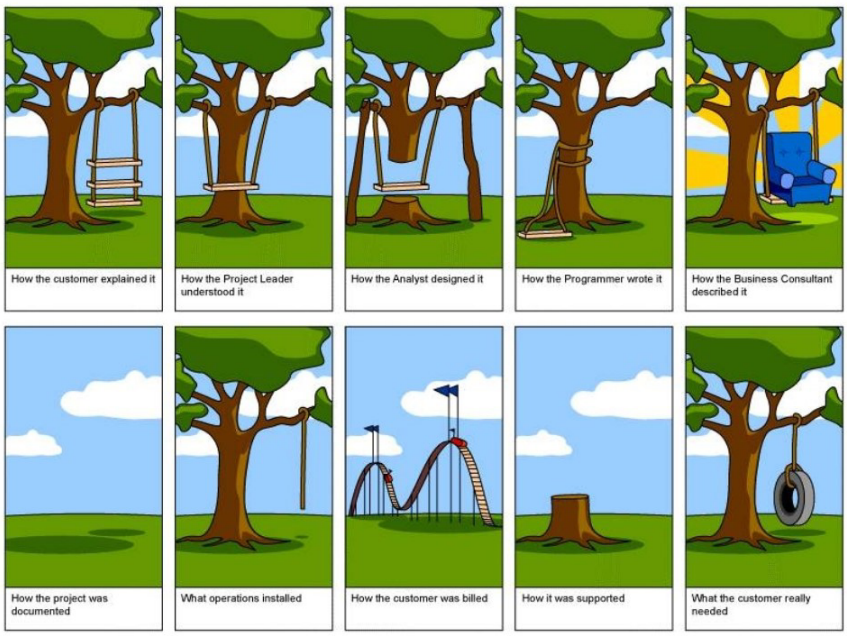

Figura 2.1: Metafora dell'altalena

L'altalena rappresentata in figura 2.1[1] è la metafora più comune per la gestione dei requisiti nei progetti software. Lo scopo che si prefigge questo documento è quello di formalizzare ciò che dovrà essere sviluppato e implementato nelle fasi successive del progetto. Per le specifiche di progettazione verranno utilizzati diagrammi UML, in quanto sono uno standard ed evitano le possibili ambiguità. [PreRog07]

2.1 L'idea

Si vuole realizzare un'applicazione *web based* che permetta a degli utenti di effettuare prenotazioni. Lo scenario tipico è quello di un'attività ristorativa che vuole memorizzare le prenotazioni dei tavoli per i propri clienti. L'utilizzatore ideale del sistema è il gestore (manager) o dipendente di un ristorante, trattoria, pizzeria o altre attività che hanno bisogno di gestire prenotazioni dei tavoli. L'applicazione deve consentire l'inserimento, la visualizzazione, la modifica e la rimozione delle prenotazioni. In particolare, tramite internet, deve essere possibile gestire le prenotazioni dei tavoli richieste dai clienti dell'utente, che di solito avviene mediante telefonata o contatto diretto con il manager dell'attività; inoltre deve consentire la consultazione dello storico delle prenotazioni.

Il nome della web-app sarà "***My Restaurant Booking Manager (MyRBM)***".

Le funzionalità richieste sono:

- **Registrazione** dell'account utente
- **Login** dell'utente nel sistema
- **Visualizzazione** delle prenotazioni
- **Prenotazione**
- **Modifica** o **rimozione** di prenotazioni create in precedenza
- **Logout** dell'utente dall'applicazione

[1] La fonte dell'immagine che rappresenta la metafora dell'altalena (Fig. 2.1) è il sito: *http://www.francescoficetola.it/2013/09/09/cosa-lutente-realmente-voleva-la-metafora-dellaltalena/*

Esempio. Un dipendente di un ristorante riceve una telefonata da parte di un cliente "Pippo", che vuole prenotare un tavolo per 4 persone a una certa data e ora, fornendo inoltre un contatto telefonico e/o indirizzo e-mail, ed eventualmente altri dettagli (note). Il dipendente verifica la disponibilità di posto per la data e ora fornita e in caso di esito positivo inserisce la prenotazione con le informazioni del cliente; la prenotazione effettuata verrà visualizzata come mostrato in figura 2.2.

Booking Time	Date	Surname	Name	Num of People	Email	Tel	Note	New Customer
10:00	02/05/2014	pluto	pippo	4	pippo.pluto@mail.com	3335689758	1 vegan	Yes

Figura 2.2: Esempio di prenotazione

2.2 Glossario

TERMINE	DESCRIZIONE	SINONIMI
User	Attore del sistema	Utente, gestore, dipendente, client, manager
Cliente	Attore esterno al sistema che interagisce con l'Utente	clienti nuovi, clienti fidelizzati
Login	Azione dell'utente per essere autenticato nel sistema	accesso al sistema, connessione, autenticazione
Logout	Azione dell'utente per uscire dal sistema	uscita dal sistema, disconnessione
Account	Informazioni dell'utente registrato nel sistema	account ristorante
Prenotazione	Inserimento delle informazioni dettagliate del cliente che prenota nel sistema	booking, reservation, promemoria
Visualizza prenotazioni	Lista delle prenotazioni sulla home page	manage reservation
Home Page	Pagina principale dell'applicazione dove è visualizzata la lista delle prenotazioni	pagina principale
Filter	Filtro che consente di visualizzare le prenotazioni di una determinata data	filtro
Password	sequenza di caratteri alfanumerici scelta dall'utente per accedere al sistema	parola segreta
MyRBM	Nome abbreviato della web-app	My Restaurant Booking Manager

2.3 Requisiti Funzionali

I requisiti funzionali descrivono le funzionalità ed i servizi forniti dal sistema (cosa deve essere fatto). Nel ciclo di sviluppo software i requisiti funzionali rappresentano i casi d'uso. Di seguito sono riportate nel dettaglio le funzionalità richieste.

Registra account

Rappresenta la pagina di benvenuto (welcome) dell'applicazione. Ogni utente che intende utilizzare l'applicazione deve essere registrato nel database. Al primo accesso l'utente per poter fare il Login nel sistema deve inserire dei dati personali di registrazione:

- indirizzo e-mail
- nome del ristorante
- password
- conferma password

Una volta che l'utente ha inserito i suoi dati, il sistema li memorizza nel database per poterli utilizzare successivamente in fase di login. Per l'autenticazione nel sistema l'utente dovrà inserire nome del ristorante come nome account e password scelta.

Login

Questa funzionalità permette al sistema di autenticare l'utente tramite l'account creato precedentemente nel caso d'uso registra account.

In fase di login il sistema mostra all'utente i campi in cui inserire nome del ristorante e password, verifica la correttezza dei dati inseriti confrontandoli con i dati presenti nel database.

Se i dati inseriti risultano corretti, l'utente viene autenticato nel sistema e può usare l'applicazione, se i dati di autenticazione non sono corretti il sistema propone all'utente di modificarli.

Visualizzazione delle prenotazioni

Questo è l'aspetto più importante dell'applicazione, ovvero la funzione principale.

La visualizzazione delle prenotazioni deve prevedere 3 viste:

1. Se non ci sono prenotazioni per il giorno attuale il sistema mostra nella home page "There are no Reservations - Data attuale"

2. Se sono presenti prenotazioni per il giorno attuale, nella home page verrà mostrata una tabella con le prenotazioni odierne. Per ogni prenotazione vengono mostrate tutte le informazioni inserite, e vengono evidenziati i clienti che non erano presenti nel database, ovvero i nuovi clienti dell'attività ristorativa.

3. Il sistema deve prevedere una sezione "Filter" da cui è possibile visualizzare la lista di prenotazioni di una data scelta dall'utente. Quindi l'utente seleziona una data e il sistema visualizza le prenotazioni relative alla stessa data se presenti, altrimenti visualizza "There are no Reservations".

Prenotazione

L'utente che vuole memorizzare una richiesta di prenotazione concorda con il cliente data e ora.

Il sistema gli presenta un form in cui inserire le informazioni della prenotazione con i seguenti campi:

- Data: la data della prenotazione
- Ora: orario della prenotazione
- Nome del cliente (facoltativo)
- Cognome del cliente: riferimento al cliente
- Numero di telefono (facoltativo): per poter eventualmente contattare il cliente
- Indirizzo e-mail: per memorizzare il cliente nel database
- Numero di persone: dato fondamentale della prenotazione

- Note (facoltativo): in questo campo sarà possibile esprimere informazioni in più sulla prenotazione, ad esempio si può inserire che nel tavolo c'è una persona celiaca oppure il manager può inserire informazioni su un cliente.

Il sistema crea l'evento alla data e ora stabilita, lo memorizza nel database e aggiorna la lista delle prenotazioni.

Modifica - Rimozione

L'utente visualizza la lista delle prenotazioni, selezionando un determinato evento l'applicazione dovrà consentire la modifica o eliminazione dello stesso.

Logout

Il sistema deve fornire la procedura di uscita dall'applicazione. Effettuando il logout l'utente viene scollegato dall'applicazione.

2.4 Requisiti non funzionali

I requisiti non funzionali non sono collegati direttamente con le funzioni implementate dal sistema, ma piuttosto alle modalità operative, di gestione. Definiscono vincoli sullo sviluppo del sistema. In seguito verranno descritti tali requisiti.

Responsive web design

Nello sviluppo dell'applicazione è richiesto l'utilizzo della tecnica di web design Responsive, in modo che le pagine web adattino automaticamente il layout per fornire una visualizzazione ottimale in funzione dell'ambiente nei quali vengono visualizzati: pc, tablet, smartphone sono i principali.

Durata sessione

Se l'utente non esegue alcuna azione nel sistema, la sessione deve interrompersi dopo 20 minuti, quindi scollegare l'utente.

Visualizzazione delle prenotazioni

Le prenotazioni vengono elencate in una lista in ordine di data e con tutte le informazioni fornite dal cliente, inoltre deve essere intuibile quali sono i nuovi clienti dell'utente.

Linguaggio di markup

Il linguaggio di markup da utilizzare è HTML5.

Database

Le informazioni da salvare nel database riguardano gli account degli utilizzatori del sistema e le prenotazioni effettuate dal sistema. Per quanto riguarda le prenotazioni devono essere memorizzate le informazioni relative al cliente.

Lingua

L'applicazione deve essere in lingua inglese.

Sicurezza

Il sistema deve prevedere delle norme di sicurezza:

- le password salvate nel database devono essere criptate
- controllare le variabili che arrivano dai form per evitare attacchi di *SQL Injection, Form Injection, Variable Injection.*

Concorrenza

L'applicazione deve prevedere l'utilizzo di più account contemporaneamente, e uno stesso account deve poter essere utilizzato da più utenti allo stesso momento.

2.5 Casi d'uso

Un caso d'uso specifica cosa ci si aspetta da un sistema ("what?") ma nasconde il suo comportamento ("how?"). E' una sequenza di azioni, con varianti, che producono un risultato osservabile da un attore e rappresenta un requisito funzionale. Ogni sequenza (detta scenario) rappresenta l'interazione di entità esterne al sistema (attori) con il sistema stesso o sue componenti. Il flusso principale degli eventi viene separato dalle varianti alternative.

L'attore rappresenta un soggetto o un'entità che non fa parte del sistema, ma interagisce in qualche modo con esso. Individua un ruolo che l'utente ricopre nell'interagire con il sistema. Gli attori eseguono i casi d'uso.

Nel nostro caso l'attore del sistema è l'utente (gestore/manager di un ristorante), i casi d'uso sono: Registra account, Login, Visualizza prenotazioni, Prenota, Modifica, Logout.

2.5.1 Diagramma dei casi d'uso

Il diagramma dei casi d'uso è il modello che descrive i requisiti del sistema in termini di funzionalità (Casi d'Uso) e ambiente circostante (Attori). Mostra **cosa** deve fare il sistema.

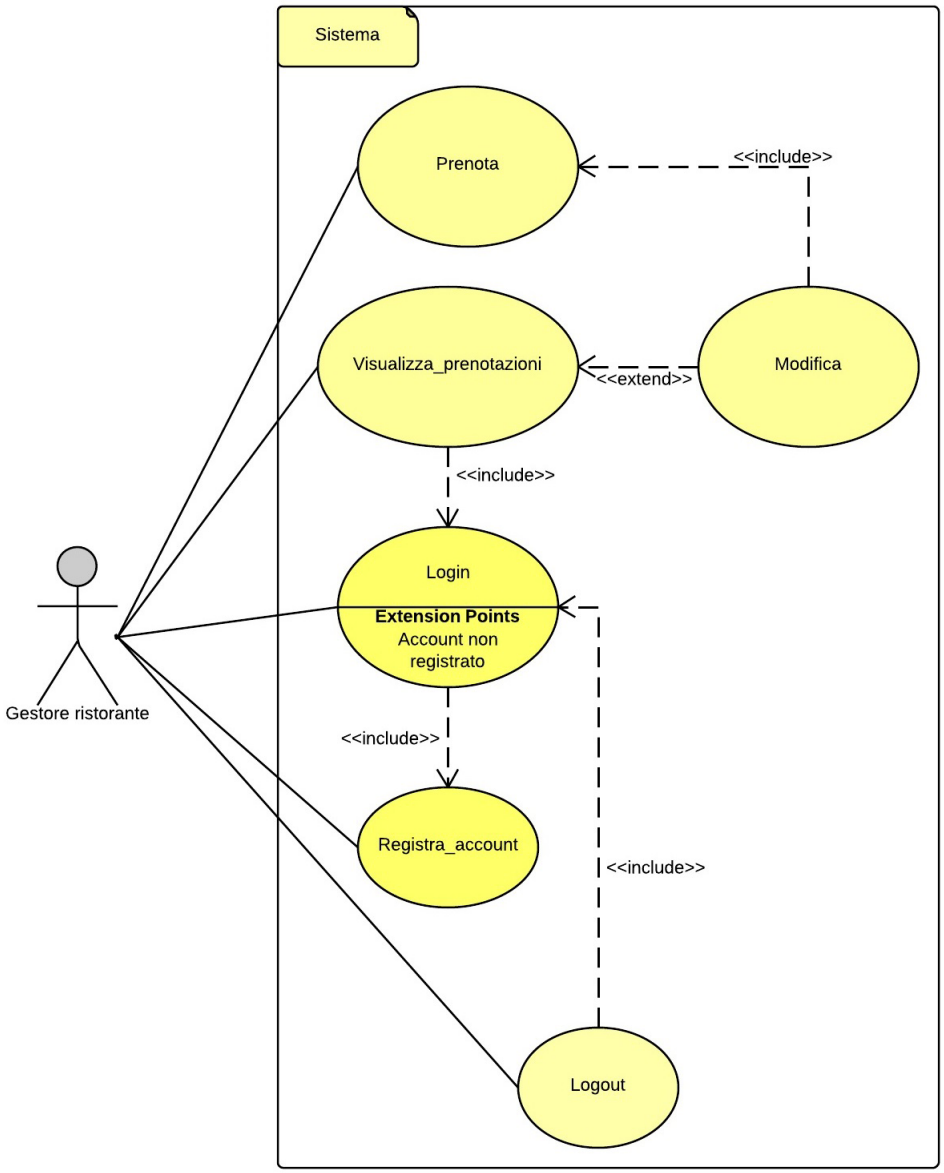

Figura 2.3: Diagramma dei casi d'uso

Di seguito vengono descritte le specifiche dei casi d'uso con scenari principali e alternativi. Uno scenario è una sequenza di passi che descrivono l'interazione tra un sistema e un attore (che dovrebbe trarre vantaggio dall'interazione).

2.5.2 Registra account

- **Nome caso d'uso**: Registra account
- **Id**: UC1
- **Attori**: Utente
- **Precondizioni**:

1. l'utente non è registrato nel sistema

- **Scenario principale**:

1. l'utente accede per la prima volta al sistema
2. il sistema visualizza la scelta tra Login e Registra account
3. l'utente sceglie Registra account
4. il sistema visualizza i campi da compilare per registrare l'account ristorante
5. l'utente inserisce i dati di registrazione
6. il sistema verifica la correttezza dei dati
   (a) se i dati non sono corretti il sistema evidenzia i campi da modificare e ritorna al punto 4
7. se i dati sono corretti il sistema conclude la fase di registrazione
8. l'utente può effettuare il login

- **Postcondizioni**:

1. L'utente è registrato nel sistema

- **Scenario secondario**:

1. l'utente accede al caso d'uso registra account

2. il sistema visualizza i campi da compilare per la registrazione dell'account
3. l'utente inserisce i dati
4. il sistema verifica che i dati sono gia presenti nel database
5. il sistema avvisa l'utente che i dati inseriti sono già stati utilizzati

- **Postcondizioni**:

1. Inizia il caso d'uso Login

2.5.3 Login

- **Nome caso d'uso**: Login
- **Id**: UC2
- **Attori**: Utente
- **Precondizioni**:

1. L'utente ha avviato il sistema

- **Scenario principale**:

1. il caso d'uso inizia quando l'utente avvia l'applicazione da un browser
2. il sistema visualizza i campi dove inserire username e password dell'utente
3. l'utente inserisce username e password
4. il sistema verifica se username e password sono corretti
   (a) se username e password non sono corretti si ritorna al punto 2.
5. Se username e password sono corretti il sistema visualizza la homepage

- **Postcondizioni**:

1. l'utente è autenticato nel sistema e visualizza la home page dell'applicazione

- **Scenario secondario**:

1. l'utente non è registrato nel sistema
2. il sistema avvia il caso d'uso Registra account

- **Postcondizioni**:

1. inizia il caso d'uso Registra account

2.5.4 Visualizza prenotazioni

- **Nome caso d'uso**: Visualizza prenotazioni
- **Id**: UC3
- **Attori**: Utente
- **Precondizioni**:

1. L'utente ha effettuato il login

- **Scenario principale**:

1. Il sistema mostra la lista delle prenotazioni del giorno attuale
2. L'utente visualizza le prenotazioni del giorno attuale

- **Postcondizioni**:

1. L'utente può effettuare modifiche alla lista prenotazioni

- **Scenario secondario 1**:

1. Il sistema mostra la lista delle prenotazioni del giorno attuale
2. L'utente vuole visualizzare la lista prenotazioni di un giorno non attuale

3. Il sistema chiede all'utente la data interessata
4. L'utente sceglie la data da visualizzare
5. Il sistema mostra la lista delle prenotazioni della data scelta

- **Postcondizioni**:

1. L'utente visualizza le prenotazioni della data scelta e può effettuare modifiche

- **Scenario secondario 2**:

1. L'utente vuole visualizzare le prenotazioni di una certa data (attuale o non), in cui ancora non esistono prenotazioni
2. Il sistema avvisa l'utente che per quella data non ci sono prenotazioni

- **Postcondizioni**:

1. L'utente visualizza che non ci sono prenotazioni per una determinata data

2.5.5 Prenota

- **Nome caso d'uso**: Prenota
- **Id**: UC4
- **Attori**: Utente
- **Precondizioni**:

1. L'utente ha effettuato il login nel sistema

- **Scenario principale**:

1. Il caso d'uso inizia quando l'utente vuole inserire una prenotazione
2. Il sistema chiede all'utente i dati della prenotazione
3. L'utente inserisce i dati richiesti

4. Il sistema verifica la correttezza dei dati

    (a) se i dati inseriti non sono corretti il sistema avvisa l'utente e ritorna al punto 2

5. se i dati sono corretti il sistema aggiunge la prenotazione

- **Postcondizioni**:

1. L'utente può visualizzare o modificare la prenotazione effettuata

2.5.6 Modifica

- **Nome caso d'uso**: Modifica
- **Id**: UC5
- **Attori**: Utente
- **Precondizioni**:

1. L'utente ha creato la prenotazione

- **Scenario principale**:

1. Il caso d'uso inizia se l'utente vuole effettuare modifiche ad una prenotazione
2. Il sistema mostra la lista delle prenotazioni
3. L'utente sceglie la prenotazione da modificare
4. Il sistema mostra i dettagli della prenotazione
5. L'utente effettua le modifiche desiderate
6. Il sistema controlla la correttezza dei dati modificati

    (a) se i dati inseriti non sono corretti il sistema avvisa l'utente e ritorna al punto 4

7. Se i dati inseriti sono corretti il sistema aggiorna la prenotazione

- **Postcondizioni**:

1. L'utente visualizza la prenotazione aggiornata

- **Scenario secondario**:

1. L'utente vuole cancellare una prenotazione
2. Il sistema mostra la lista delle prenotazioni
3. L'utente sceglie la prenotazione da eliminare
4. Il sistema elimina la prenotazione scelta e aggiorna la lista prenotazioni

- **Postcondizioni**:

1. L'utente visualizza la lista delle prenotazioni aggiornata

2.5.7 Logout

- **Nome caso d'uso**: Logout
- **Id**: UC6
- **Attori**: Utente
- **Precondizioni**:

1. L'utente ha effettuato il login

- **Scenario principale**:

1. il caso d'uso inizia quando l'utente vuole uscire dal sistema
2. l'utente chiede al sistema di disconnettersi
3. il sistema disconnette l'utente

- **Postcondizioni**:

1. L'utente è disconnesso dal sistema

2.6 Diagramma delle attività

Il diagramma delle attività, in UML, è un diagramma di flusso (con alcuni elementi aggiuntivi) che mostra una sequenza di attività. Viene utilizzato per rappresentare i passi (le transazioni) che compongono il flusso di un caso d'uso, descrivono quindi il comportamento dinamico di un sistema. La rappresentazione delle attività è comoda in quanto consente di rappresentare sinteticamente flusso principale e flussi alternativi. Il diagramma in figura 2.4 mostra il flusso delle attività dell'utente per visualizzare, aggiungere o modificare prenotazioni.

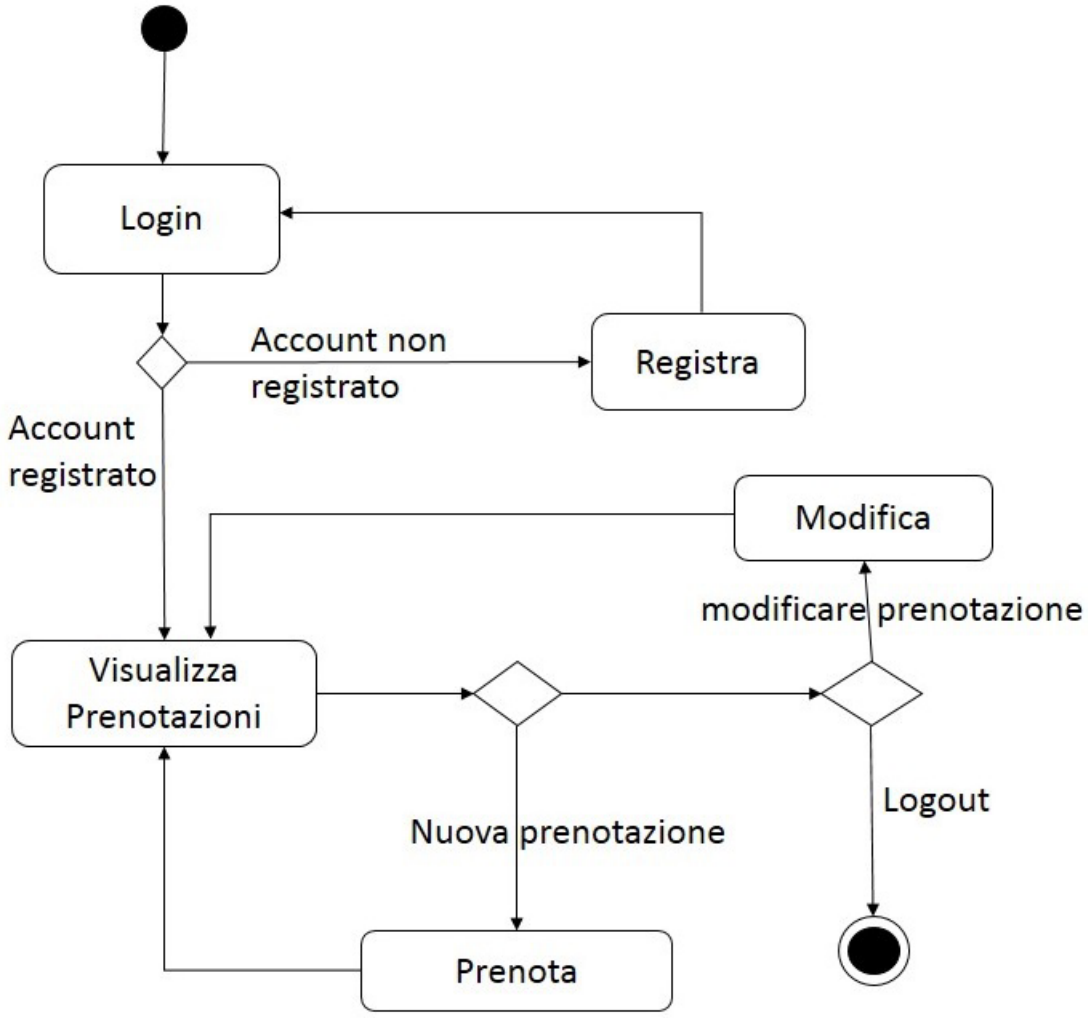

Figura 2.4: Diagramma delle attività

Capitolo 3

Progettazione

In questa fase si definisce l'architettura software su cui si baserà la realizzazione del prodotto ***My Restaurant Booking Manager***. Ho scelto una strutturazione tipica su tre livelli logico-funzionali (applicazioni *Three-Tier*) ma che possono essere distribuiti anche su più li velli (applicazioni *Multi-Tier*)

L'applicazione *Web Three-Tier* si sviluppa su 3 livelli [PMA13]:

- Presentazione (Cliente): costituisce l'interfaccia utente dell'applicazione e corrisponde al browser Web. Le tecnologie scelte da utilizzare sono HTML (formato per la presentazione di ipertesti) e CSS (foglio di stile per documenti HTML).

- Applicazione (Servente Web): è il livello logico della web-app, la componente elaborativa. Dal lato server (Server-side) utilizza la tecnologia PHP, dal lato client(Client-side) utilizza il linguaggio di scripting Javascript.

- Dati (Servente RDBMS): consente di modellare e gestire il contenuto informativo dell'applicazione. La tecnologia usata a questo livello è il Database relazionale MySQL.

La figura 3.1[1] rappresenta graficamente la struttura di una web application:

[1]Fonte dell'immagine: *"scuola.linux.it/docs/altre_ scuole/planck/tecnologie-web/tecnologie-web12.html"*

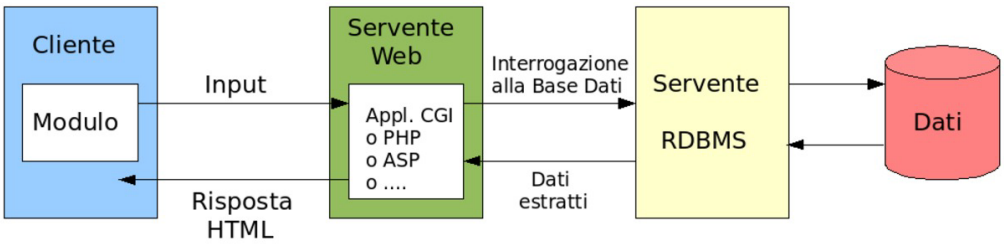

Figura 3.1: Struttura web application

Gli strumenti che ho scelto di utilizzare sono:

XAMPP che è una piattaforma software open source costituita dal server web **Apache**, il database **MySQL**, un interprete **PHP**5 e l'applicazione "**phpMyAdmin**" per la gestione via web delle basi di dati, cui si accede collegandosi all'indirizzo "http://localhost/mysql".

Gli script eseguiti nella parte Client-side sono scritti nel linguaggio **Javascript** e qualche funzione dalla libreria **jQuery**. **HTML** per la visualizzazione delle pagine al client che vengono create dinamicamente dalle pagine PHP che risiedono nel Server.

Per la stesura del codice PHP ho utilizzato l'ambiente di sviluppo open source **Eclipse** per PHP.

XAMPP è facile da usare rispetto ad altri software simili, basta scaricarlo gratuitamente e installarlo. Una volta installato XAMPP bisogna caricargli le pagine PHP, gli script Javascript ed eventuali fogli di stile **CSS** ed è possibile avviare il server Apache e il software incluso MySQL. Quando il server è attivo è possibile gestire il database tramite l'interfaccia grafica "phpMyAdmin".

Per effettuare i test ho usato l'indirizzo locale *"http://localhost/myrbm"* dove "myrbm" è il nome del package contenti tutti i file necessari; quindi da un qualsiasi browser è possibile testare il funzionamento del codice inserito collegandosi all'indirizzo impostato su XAMPP.

3.1 Tecnologie utilizzate

In questa sezione sono descritte in dettaglio le tecnologie utilizzate per la realizzazione dell'applicazione. Lo studio di esse ha occupato una parte rilevante del mio lavoro in quanto sono tecnologie di nuova generazione.

HTML

HTML (Hypertext Markup Language - linguaggio di "marcatura" di Ipertesti*)* è il linguaggio per creare pagine ipertestuali (pagine web). HTML non è come un linguaggio di programmazione ma un semplice sistema di contrassegno, i cui TAG vengono riconosciuti ed interpretati dai browser web. Questo potente mezzo di comunicazione consente di visualizzare i contenuti delle pagine nella veste grafica preferita e permette l'introduzione di elementi multimediali (suoni, immagini, filmati ecc.) nonché la consultazione di documenti in modo non sequenziale. I documenti HTML sono spesso chiamati "Pagine Web", in pratica un file in formato HTML è un file che, oltre a contenere il testo che verrà visualizzato dal browser, contiene anche dei comandi (racchiusi sempre tra i simboli "<" e ">" chiamati "***tag***") che associano al testo un particolare attributo. Gli elementi del linguaggio vengono detti *mark up tag* o semplicemente *tag*: essi di solito sono utilizzati a coppie, e possono contenere uno o più attributi. Generalmente l'aspetto di un tag è il seguente

<nome tag> Testo influenzato dal tag </nome tag>

Una pagina in codice HTML può essere redatta con un semplice editor di testi e salvata con estensione .html o .htm. Quando il browser (Mozilla Firefox, Google Chrome ecc.) carica un file HTML, legge e interpreta i tag in esso contenuti e presenta il risultato di tale elaborazione sullo schermo. Poiché i file sono soggetti all'interpretazione del browser può succedere che alcune parti del testo o alcune sue particolari formattazioni possano essere interpretate in modo diverso da browser differenti.

JAVASCRIPT

Javascript è un linguaggio di scripting per applicazioni client, server che aggiunge elementi interattivi alle pagine web (HTML) con la possibilità di interfacciarsi a database o di gestire i file. Linguaggio di scripting sta ad indicare che i programmi creati in Javascript sono interpretati e non compilati, quindi non possono essere eseguiti direttamente dal sistema operativo, ma è necessario disporre di un *browser* che possa interpretare ed eseguire le istruzioni. L'utilizzo principale di Javascript riguarda l'ottimizzazione di pagine HTML, per creare pagine dinamiche e interattive. Javascript è un linguaggio dinamico orientato agli oggetti; esso ha tipi e operatori, oggetti nucleo, e metodi. La sua sintassi deriva dai linguaggi Java e C, quindi molte strutture da questi linguaggi ricorrono anche in Javascript. Per poter scrivere codice Javascript è sufficiente un editor di testo da salvare con estensione ".js" per poter essere richiamato nelle pagine HTML, oppure si può essere inserire codice Javascript direttamente all'interno dei file HTML utilizzando gli opportuni tag: *<script type="text/javascript">codice Javascript</script>*.

CSS

Nel progetto sono stati utilizzati i *CSS* o fogli di stile a cascata (dall'inglese Cascading Style Sheet). Essi sono un insieme di regole redatte dal *W3C* (*World Wide Web Consortium*) per definire l'aspetto delle pagine *HTML* e *XHTML*. La loro caratteristica fondamentale è la possibilità di separare i contenuti dalla formattazione e imporre una programmazione più chiara e facile da utilizzare, sia per l'autore che per l'utente. Una pagina web è formata fondamentalmente da due elementi: i contenuti veri e propri che la pagina intende fornire e la formattazione , cioè l'aspetto con cui i contenuti saranno mostrati all'utente.

PHP

PHP (acronimo ricorsivo di *Hypertext Prepocessor*) è un linguaggio di scripting interpretato (non compilato) server-side, con licenza open source, originariamente concepito per la realizzazione di pagine Web dinamiche. Attualmente è utilizzato principalmente per sviluppare applicazioni Web. Il codice PHP viene usato per ge-

nerare dinamicamente i documenti HTML che il client deve ricevere e visualizzare nel browser, ad esempio quando il server web Apache deve gestire un documento richiesto con estensione "*.php*" utilizza il processore PHP che restituisce alla fine dell'elaborazione un documento HTML.

I file PHP sono strutturati come un documento HTML, ma contengono sezioni di codice PHP delimitate da *<?php* e *?>*; l'interprete effettua il parsing del file e sostituisce le sezioni di codice PHP con il codice HTML risultante dall'esecuzione. L'output di PHP è un documento HTML, il client riceve tale output senza vedere il codice PHP che lo ha generato.

Per l'accesso al database MySQL PHP mette a disposizione molte funzioni, quali ad esempio *mysql_ connect(), mysql_ select_ db(), mysql_ query()* ecc.

XAMPP

XAMPP è una piattaforma software gratuita costituita da *Apache Http Server*, un database *MySQL* e tutti gli strumenti necessari per usare i linguaggi di programmazione *PHP* e *Perl*. Il nome XAMPP è un acronimo dove

X = cross-platform (multipiattaforma)

A = Apache (server HTTP)

M = MySQL (database)

P = PHP (interprete)

P = Pearl (interprete)

Ho scelto di usare questo software per creare il mio web server per la sua semplicità di installazione e utilizzo, soprattutto per creare siti da testare in locale prima di renderli pubblici sulla rete; inoltre la comodità di XAMPP sta anche nel fatto che molte sue funzioni possono essere intuitivamente configurate via web con un browser grazie alla funzione *phpMyAdmin*. L'interfaccia principale del programma (Pannello di Controllo) è descritta in figura 3.2.

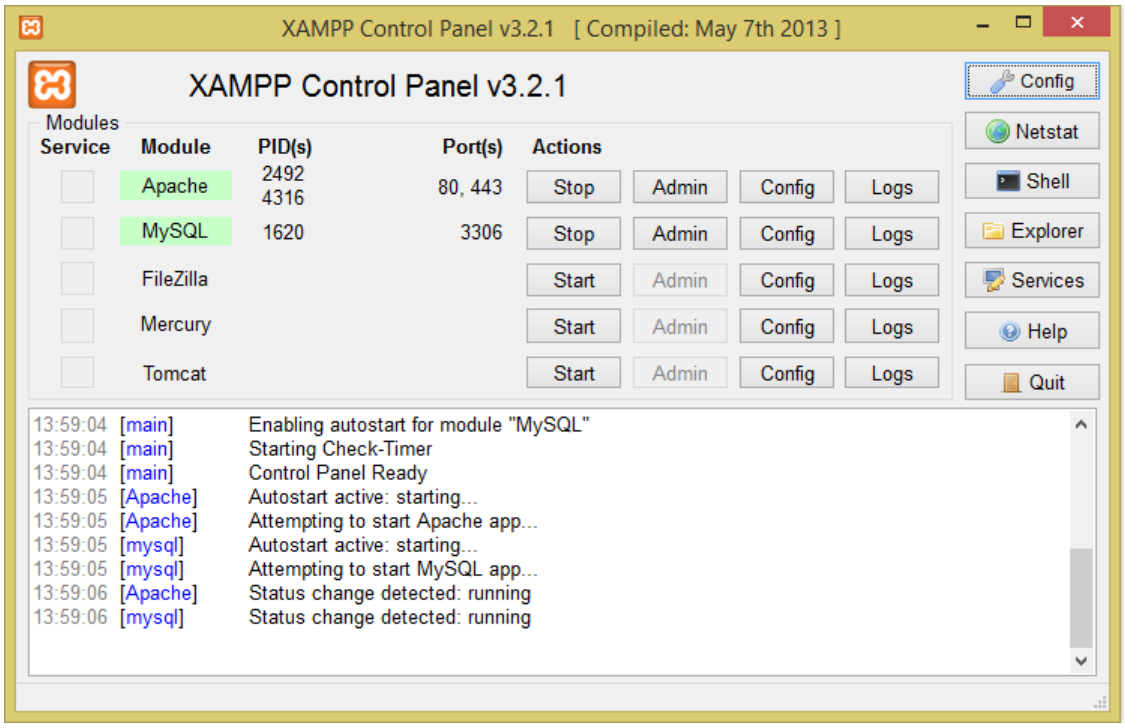

Figura 3.2: Pannello di controllo XAMPP

WEB SERVER APACHE

Apache (o meglio *Apache HTTP Server*) è una piattaforma server web modulare in grado di operare nei sistemi operativi più diffusi. Le pagine **.php** contengono dei codici destinati a produrre dei comportamenti e a generare dinamicamente conenuti HTML, perchè ciò sia possibile è necessaria la mediazione di un web server. Un *web server* è un programma che si occupa di ascoltare un canale di comunicazione per intercettare una richiesta da servire. Il client, utilizzando un browser, invia un messaggio di richiesta HTTP , contenente la URL, attraverso il collegamento di rete al web server; questo, catturata la richiesta, risponde, sempre attraverso il protocollo HTTP , con una pagina HTML con il contenuto informativo desiderato dal client. L'insieme dei web server presenti su internet forma il **WWW** ossia il ***W****orld* ***W****ide* ***W****eb*, il servizio piu sfruttato della rete.

Il web server Apache presenta un'architettura modulare, quindi ad ogni richiesta del client, vengono svolte funzioni specifiche da ogni modulo di cui è composto,

come u nità indipendenti. Ciascun modulo si occupa di una funzionalità, ed il controllo è gestito dal core[2]:

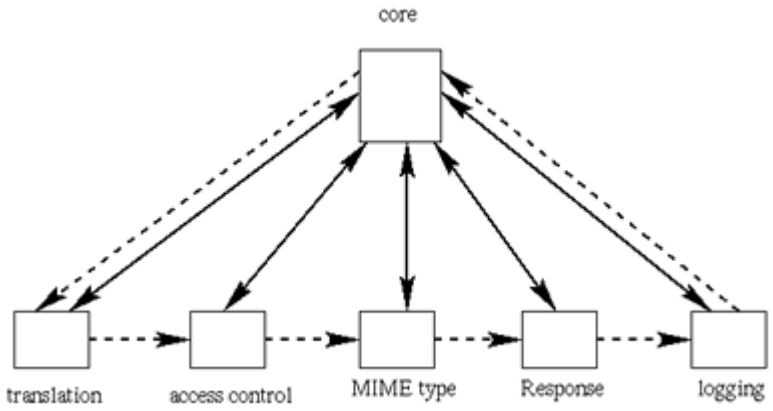

Figura 3.3: Architettura modulare di Apache web server

Le linee continue dello schema rappresentano il flusso dei dati reale, le linee tratteggiate il flusso dei dati astratti che formano la pipeline. I moduli che compongono il web server Apache sono:

Core: programma principale composto da un ciclo sequenziale di chiamate ai moduli.

Translation: traduce la richiesta del client.

Access control: controlla eventuali richieste malevoli.

MIME Type: verifica il tipo di contenuto.

Response: invia la risposta al client e attiva eventuali procedure.

Logging: tiene traccia delle operazioni che sono state eseguite.

[2]Fonte *Wikipedia* all'indirizzo: *http://it.wikipedia.org/wiki/File:Apache_architettura.png*

Il core suddivide la richiesta sequenzialmente ai vari moduli, usando i parametri di uscita di un modulo come parametri di accesso per l'altro modulo, creando così l'illusione di una comunicazione orizzontale fra i moduli (*pipeline software*). Sopra il ciclo core c'è un ulteriore ciclo di polling svolto da un demone che interroga continuamente le linee logiche da cui possono pervenire messaggi di richiesta.

MYSQL

Innanzitutto una base di dati (*database*) è un insieme di dati logicamente correlati fra loro. I *Data Base Management System (DBMS)* sono prodotti software in grado di gestire database che hanno grandi quantità di dati, che condividono i dati fra più utenti e applicazioni e che utilizzano dei sistemi di protezione e autorizzazione per l'accesso ai dati. Esistono diversi tipi di DBMS: gerarchico, reticolare, relazionale, ad oggetti; il modello che più si adatta alle mie esigenze è il modello relazionale che organizza i dati in tabelle, basandosi sulle relazioni fra essi.

MySQL è un sistema di gestione di basi di dati relazionali multi-piattaforma distribuito dalla compagnia svedese '*MySQL AB*' come software libero sotto licenza *GPLV2* (*General Public License version 2*). Più precisamente *MySQL* è un *RDBMS* ("*Relational DataBase Management System*"), ossia un sistema di gestione per database relazionali, che si basa sul linguaggio ***SQL*** ("*Structured Query Language*" è il linguaggio standard di interrogazione dei database). MySQL si occupa della strutturazione e della gestione a basso livello dei dati stessi, in modo da velocizzarne l'accesso, la modifica e l'inserimento di nuovi elementi.

Software utilizzati

- *Windows 8.1 pro x64*: sistema operativo
- *XAMPP*: web server
- *Eclipse*: ambiente di sviluppo (IDE) open source
- *ArgoUml*: per la realizzazione dei diagrammi UML
- *Lyx*: editor grafico per il linguaggio LaTex utilizzato per la stesura di questo documento

- *Lucidchart*: per la realizzazione del diagramma dei casi d'uso
- *Notepad++*: editor di testo
- *Mozilla Firefox, Google Chrome, Opera, Internet Explorer*: browser usati per ricerca, sviluppo e test.

3.2 Database

Quando i dati sono molti e salvarli su filesystem risulta inefficiente bisogna usare il supporto di una base di dati [Apa].

L'analisi dei requisiti ha portato all'individuazione di due entità fondamentali: "Account" e "Reservation". Le entità e le associazioni che intercorrono tra esse sono rappresentate nel seguente schema Entity/Relationship.

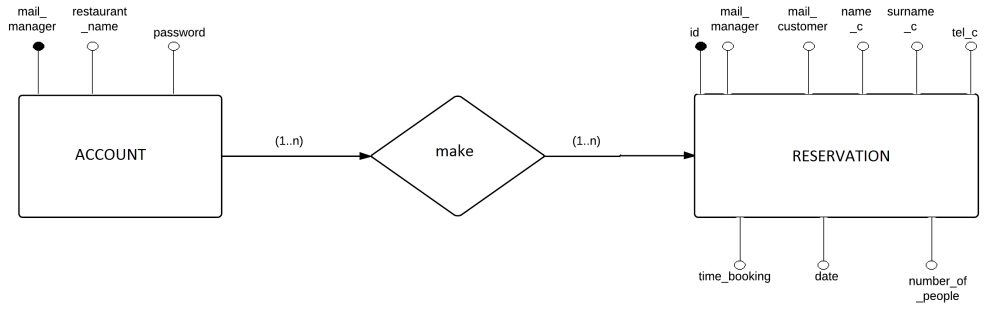

Figura 3.4: Schema Entity/Relationship

Lo schema logico ottenuto è il seguente:

ACCOUNT (email_manager, restaurant_name, password)

RESERVATION (id_reservation, email_manager, email_customer, name_c, surname_c, tel_c, time_booking, date, number_of_people)

ACCOUNT_MAKE_RESERVATION (mail_manager, id_reservation)

Nelle sezioni seguenti verrà approfondita l'analisi delle entità , precisando il ruolo di ciascuna.

3.2.1 Entità "Account"

L'entità "Account" serve per tenere traccia di tutti gli utilizzatori del sistema (manager ristorante). Il compito principale di "Account" è la gestione delle credenziali per poter effettuare il login tramite e-mail e password scelti dall'utente in fase di registrazione.

Gli attributi di Account sono i seguenti:

e-mail_manager serve a identificare univocamente un utente del sistema (primary key)

restaurant_name serve per visualizzare il nome dell'attività ristorativa gestite tramite il sistema.

password utilizzata in fase di login insieme all' e-mail per autenticare l'utente nel sistema. Questo campo per motivi di sicurezza viene salvato nel database criptato.

L'entità "account" risulta la seguente:

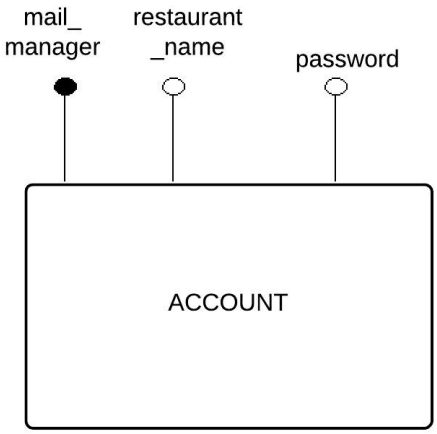

Figura 3.5: Entità account

Un esempio di tabella Account è rappresentato in figura 3.6.

Mail manager (primary Key)	Restaurant name	password
esempio@mail.com	Restaurant	12345

Figura 3.6: Esempio di tabella Account

3.2.2 Entità "Reservation"

L'entità "Reservation" tiene traccia di tutte le prenotazioni effettuate dagli utenti.

Gli attributi di Reservation sono i seguenti:

id_reservation serve ad identificare univocamente la prenotazione

email_manager utilizzato per associare la prenotazione ad un determinato utente

email_customer serve ad identificare il cliente[3] che ha richiesto la prenotazione

name_c nome del cliente

surname_c cognome del cliente

tel_c numero di telefono del cliente

time_booking ora della prenotazione

date data della prenotazione

number_of_people specifica il numero di posti a sedere che si vuole prenotare

L'entità "reservation" risulta quindi la seguente:

[3]il cliente è un agente esterno al sistema da non confondere con l'utente

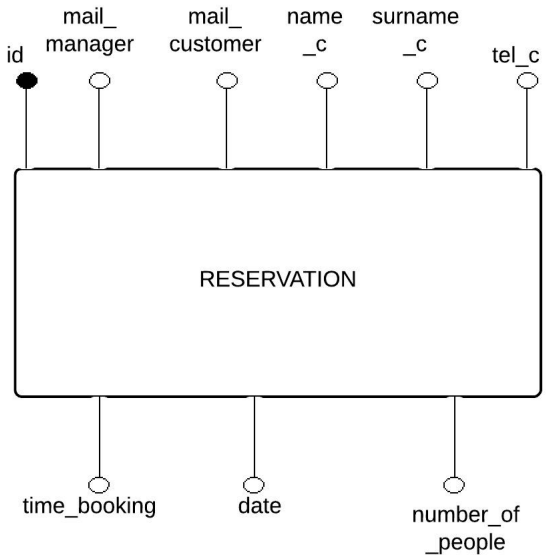

Figura 3.7: Entità reservation

Un esempio di tabella Reservation è rappresentato in figura 3.8.

Id (Key)	Mail manager	Mail customer	Name c	Surname c	Tel c	Time booking	Date	Number of people
01	esempio@ mail.com	customer@mail .it	Mario	Rossi	3335869775	13:00	21/03/2014	4

Figura 3.8: Esempio di tabella Reservation

3.2.3 Relazione "make/effettua"

La relazione "make" è intesa come "effettua" una o più prenotazioni. Ogni account è associato a uno o molte prenotazioni; ogni prenotazione è associata a un account;

- Entità: Account, Prenotazione
- Attributi: email_manager, id_reservation

3.2.4 Creazione database

Il database utilizzato è MySQL. La creazione delle tabelle che serviranno all'applicazione è avvenuta nel seguente modo:

1. L'entità "Account" viene tradotta nella tabella "account", con inserimento di dati per test

```
-- Struttura della tabella `account`
--

CREATE TABLE IF NOT EXISTS `account` (
  `email_manager` varchar(50) NOT NULL,
  `name_restaurant` varchar(50) NOT NULL,
  `password` varchar(40) NOT NULL,
  PRIMARY KEY (`email_manager`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

--
-- Dump dei dati per la tabella `account`
--

INSERT INTO `account` (`email_manager`, `name_restaurant`, `password`) VALUES
('manager@email.com', 'RestaurantOne', '9b207167e5381c47682c6b4f58a623fb'),
('managertwo@two.it', 'RestaurantTwo', '3528579414912ea7816656ea7b96b29e');
```

Figura 3.9: creazione tabella account SQL

2. L'entità "Reservation" viene tradotta nella tabella "reservation", con inserimento di dati per testare il corretto funzionamento

```sql
-- Struttura della tabella `reservation`
--

CREATE TABLE IF NOT EXISTS `reservation` (
  `id_reservation` int(11) NOT NULL,
  `email_manager` varchar(50) NOT NULL,
  `email_customer` varchar(50) NOT NULL,
  `name_c` varchar(30) NOT NULL,
  `surname_c` varchar(50) NOT NULL,
  `tel_c` varchar(50) NOT NULL,
  `time_booking` int(11) NOT NULL,
  `date` varchar(11) NOT NULL,
  `number_of_people` int(11) NOT NULL,
  `note` text NOT NULL,
  `new_c` varchar(10) NOT NULL,
  PRIMARY KEY (`id_reservation`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

--
-- Dump dei dati per la tabella `reservation`
--

INSERT INTO `reservation` (`id_reservation`, `email_manager`, `email_customer`, `name_c`, `surname_c`, `tel_c`,
                           `time_booking`, `date`, `number_of_people`, `note`, `new_c`) VALUES
(4, 'manager@email.com', 'pippo@mail.com', 'Pippo', 'Brown', '3332425266', 38100, '02/02/2014', 4, '', ''),
(6, 'managertwo@two.it', 'john@email.it', 'John', 'Black', '31119181', 50400, '02/20/2014', 2, '', ''),
(7, 'managertwo@two.it', 'mark@karm.it', 'Mark', 'Green', '3359887766', 46800, '02/16/2014', 3, '', 'Yes'),
(9, 'manager@email.com', 'r.bianchi@mail.it', 'Giuseppe', 'Bianchi', '3224466890', 4260, '02/02/2014', 4, 'vegan', 'Yes'),
(10, 'manager@email.com', 'pippo@mail.com', 'Pippo', 'Brown', '', 51060, '03/02/2014', 4, '', ''),
(11, 'manager@email.com', 'jack@mail.com', 'Jack', 'White', '', 40260, '03/02/2014', 6, '', ''),
(12, 'manager@email.com', 'jack@mail.com', '', 'White', '', 33300, '03/02/2014', 3, '', ''),
(13, 'manager@email.com', 'lin@yahoo.cn', 'Lin', 'Ling', '', 43920, '03/02/2014', 2, '', 'Yes'),
(14, 'managertwo@two.it', 'john@email.it', 'John', 'Black', '', 73800, '03/02/2014', 3, '', '');
```

Figura 3.10: creazione tabella reservation SQL

Capitolo 4

Sviluppo e test

È la fase realizzativa. Ogni modulo, come unità indipendente, viene implementato e controllato per assicurarne la correttezza [Xam].

4.1 Struttura del sito MyRBM

Ogni pagina è formata da un menu che si trova in posizione fissa nella colonna a sinistra (Fig. 4.1). Ogni elemento del menu è costituito da una pagina *PHP*:

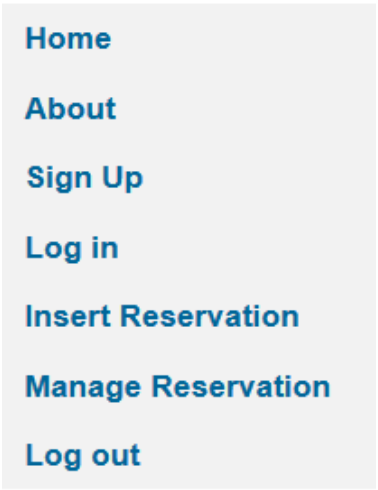

Figura 4.1: Menu di navigazione del sito

- **Home**: home page del sito che dopo il login rimanda l'utente alla pagina *Manage Reservation*;
- **About**: pagina che descrive le informazioni dell'applicazione;
- **Sign Up**: pagina di benveuto che consente la registrazione al servizio;
- **Log in**: pagina di autenticazione;
- **Insert Reservation**: è costituita da un form che riceve le informazioni della singola prenotazione da inserire nel database;
- **Manage Reservation**: pagina principale che consente la visualizzazione delle prenotazioni del giorno in corso, oppure di una data scelta dall'utente, consente inoltre la modifica o rimozione di una prenotazione tramite i pulsanti "*Modify*" e "*Cancel*" posti di fianco ad ogni prenotazione;
- **Log out**: pagina che scollega l'utente dalla sessione.

I file sorgenti (php, js, css) prodotti sono i seguenti:

about.php: pagina descrittiva dell'applicazione

home.php: pagina principale della web-app

index.php: pagina di gestione dei cookie di sessione

insertreservation.php: pagina che effettua le prenotazioni

login.php: pagina di gestione login

logout.php: pagina che scollega l'utente dalla sessione

managereservation.php: pagina di visualizzazione e gestione delle prenotazioni

modifyreservation.php: si occupa di modificare le informazioni del database

nb_function.php: classe di funzioni comuni a tutte le altre classi (pattern Singleton)

signup.php: pagina di registrazione alla web-app

functions.js: script per la gestione dei cookie

validate.js: insieme di script per i controlli di sicurezza dei dati inseriti nel sistema

myrbm.css: foglio di stile del sito

myrbm.sql: contiene il codice SQL per la creazione e l'inserimento delle tabelle nel database.

La figura 4.1 mostra la struttura grafica dei file prodotti con l'*IDE Eclipse*.

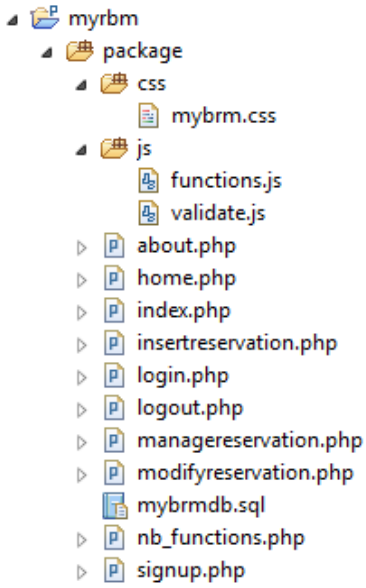

Figura 4.2: Struttura delle classi da Eclipse

4.1.1 Funzioni rilevanti per la web-app

Quando si realizzano applicazioni Web è molto utile definire una volta per tutte alcune caratteristiche comuni a tutte le pagine del sito oppure porzioni di codice da utilizzare in punti diversi dell'applicazione stessa; I file sorgenti "*index.php*", "*nb_function.php*", "*function.js*", "*validate.js*" e "*myrbm.css*" sono stati creati per essere utilizzati nelle altre classi php. In seguito vengono forniti dettagli implementativi di suddette classi.

4.1.1.1 Index (PHP) e functions (Javascript)

Il file *"index.php"* è la pagina di default richiamata quando viene digitata la *URL* o una specifica pagina del sito web; verifica se nel browser che si sta utilizzando sono abilitati i *cookies* (altrimenti invita l'utente ad abilitare i *cookies* nel proprio browser per poter utilizzare l'applicazione) e fa un redirect alla *Home page (home.php)*.

I **cookies** sono dei piccoli file di testo che sono generati e letti sul lato server sul quale è posto il sito web, e che vengono creati e memorizzati sul lato client ottimizzando in tal modo le risorse di memoria sul server.

Il codice Javascript è il seguente:

```
<script type="text/javascript">
createCookie("test", "1");
var res = getCookie("test");
if(res==1)
    window.location.href = "home.php";
else
   alert("This page needs cookies enablet to work.");
</script>
```

Figura 4.3: codice Javascript cookies

I metodi per creare il cookie (*createCookie*) e per richiamarlo (*getCookie*) sono in un file Javascript chiamato "*functions.js*" che viene utilizzato nelle pagine *php server side*:

```javascript
function createCookie(name, value) {
    var date = new Date();
    date.setTime(date.getTime() + (1 * 24 * 60 * 60 * 1000));//supportIE
    //date.setTime(date.getTime() + (date.getFullYear+1));  // not IE
    var expires = '; expires=' + date.toGMTString();
  document.cookie = name + '=' + value + expires + '; path=/';
}
function getCookie(c_name){
  if (document.cookie.length > 0)
  {
    var c_start = document.cookie.indexOf(c_name + '=');
    if (c_start != -1) {
      c_start = c_start + c_name.length + 1;
      var c_end = document.cookie.indexOf(';', c_start);
      if (c_end == -1) {
        c_end = document.cookie.length;
      }
      return unescape(document.cookie.substring(c_start, c_end));
    }
  }  return "";
}
```

Figura 4.4: funzioni Javascript per creare i cookies

4.1.1.2 Validate (Javascript)

Il file script "*validate.js*" contiene una serie di funzioni in Javascript per effettuare controlli sugli input dal lato client. Nelle pagine dove vengono immessi dati da elaborare, vengono effettuati controlli sulla correttezza delle informazioni inserite per ogni tipo di campo disponibile. Se dopo il controllo, i dati immessi risulano corretti viene visualizzato in HTML la dicitura "*ok*" accanto il campo compilato, ad esempio nel campo *"e-mail"* deve esserci un indirizzo e-mail valido per essere accettato:

Figura 4.5: esempio controllo campo input valido

se l'indirizzo inserito non è corretto viene visualizzato *"Invalid email"*:

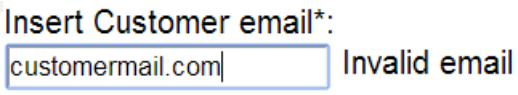

Figura 4.6: esempio controllo campo input non valido

Nelle pagine di inserimento dati che vengono salvati nel database, solo se tutti i campi inseriti risultano "*valid*" si abilita il pulsante "*Submit*", altrimenti rimane disabilitato al fine di salvaguardare la sicurezza dei dati trasmessi al server. In particolare, mediante funzioni offerte dalle api PHP (*sanitize*) vengono svolti due controlli: il primo si occupa dell'individuazione (e la successiva rimozione) dei caratteri speciali di *MySQL* (apice singolo, apice doppio, backslash, cancelletto, ecc.), mentre il secondo verifica che non vi siano dei tag HTML all'interno delle stringhe immesse nei campi di input. In questo modo un utente che vuole inserire codice malevolo nei campi di input viene bloccato.

I tipi di campi presenti nella *web app* lato client sono:

- String: l'inserimento di semplice testo (campi "*restaurant name*", "*password*") viene gestito dallo script "*validateString()*" che effettua un controllo sulla stringa inserita e visualizza "*ok*" oppure "*Invalid*".

- E-mail Sign: il controllo di un indirizzo e-mail per registrare l'utente (*Sign up*) avviene grazie alla funzione "*validateEmail_Sign()*" che stabilisce se l'email inserita è "*ok*" oppure "*Invalid email*" in questo modo:

```
function validateEmail_Sign(){
        var email = document.getElementById('email').value;
        if(email)
        {
          if(email.length <= 0 )
          {
                 document.getElementById('emailLabel').innerHTML = 'Invalid email';
                 return;
          }
         var re = /^(([^<>()[\]\\.,;:\s@\"]+(\.[^<>()[\]\\.,;:\s@\"]+)*)|" +
                "(\".+\"))@((\[[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\" +
                "])|(([a-zA-Z\-0-9]+\.)+[a-zA-Z]{2,}))$/;
          if(re.test(email))
             document.getElementById('emailLabel').innerHTML = 'ok';
          else
             document.getElementById('emailLabel').innerHTML = 'Invalid email';
        }
        else
         document.getElementById('emailLabel').innerHTML = 'Invalid email';
         if(is_Ok_Sign())
             document.getElementById('submit').disabled=false;
          else
             document.getElementById('submit').disabled=true;
}
```

Figura 4.7: funzione JS per validare email

- *is_ ok_ Sign()* è la funzione che abilita il tasto "*submit*" se i campi "*restaurant name*", "*e-mail*" e "*password*" risultano corretti, altrimenti il tasto rimane disabilitato:

```
function is_Ok_Sign()
{

    if( document.getElementById('restaurantNameLabel').innerHTML!='ok')
           return false;
    if( document.getElementById('emailLabel').innerHTML!='ok')
           return false;
    if( document.getElementById('passNameLabel').innerHTML!='ok')
           return false;

    return true;
}
```

Figura 4.8: funzione JS che abilita il tasto submit

Le altre funzioni effettuano il controllo dei dati inseriti nel form "*insert reservation*", tramite le seguenti funzioni:

- *validateName():* controlla la correttezza del campo "*surname*" (il cognome del cliente);

- *validateEmail()*: simile alla funzione "*validateEmail_Sign()*" descritta precedentemente;

- *validateTime(mess)*: controlla il formato dell'orario inserito per la prenotazione che deve essere "hh:mm" per essere accettato;

- *validateNum(mess)*: viene usato nel campo "*num_of_people*" per controllare che venga inserito un numero;

- *validateDate()*: verifica se il formato della data inserita è corretto, cioè nel formato (inglese) "*mm/gg/aaaa*" (diverso dal formato italiano "*gg/mm/aaa*"):

```
195 function validateDate(){
196         var date = document.getElementById('date').value;
197         if(xNum)
198         {
199           if(xNum.length <= 0 )
200           {
201               document.getElementById('dateLabel').innerHTML = 'Invalid Date';
202               return;
203           }
204           var re = /^\d{2}[/]\d{2}[/]\d{4}$/;
205           if(re.test(xNum))
206              document.getElementById('dateLabel').innerHTML = 'ok';
207            else
208               document.getElementById('dateLabel').innerHTML = 'Invalid Date';
209          }
210          else
211           document.getElementById('dateLabel').innerHTML = 'Invalid Date';
212
213         if(is_Ok_Insert())
214               document.getElementById('submit').disabled=false;
215            else
216               document.getElementById('submit').disabled=true;
217 }
```

Figura 4.9: funzione JS per la validazione della data

- *is_ok_Insert()*: in fine viene effettuata la verifica sui campi obbligatori per valutare se abilitare o meno il tasto "*submit*":

```
233 function is_Ok_Insert(){
234     // AddBookingTimeLabel  custAddNameLabel custSurAddNameLabel telNumLabel emailLabel pec
235
236     if( document.getElementById('AddBookingTimeLabel').innerHTML!='ok')
237             return false;
238     if( document.getElementById('custSurAddNameLabel').innerHTML!='ok')
239             return false;
240     if( document.getElementById('emailLabel').innerHTML!='ok')
241             return false;
242     if( document.getElementById('peopleNumLabel').innerHTML!='ok')
243             return false;
244
245     return true;
246 }
247
```

Figura 4.10: funzione JS per validare i campi "insert"

4.1.1.3 Functions (PHP)

La classe "*nb_functions.php*" gestisce le connessioni ed è stata implementata utilizzando il design pattern ***Singleton***. Nell'ingegneria del software il pattern Singleton è utilizzato per limitare ad una sola le istanze di un oggetto. Questo è utile quando è necessario avere esattamente un'unica istanza di un oggetto per coordinare le azioni all'interno del proprio sistema, oppure quando si hanno degli aumenti di efficienza condividendo una sola istanza. L'utilizzo del pattern Singleton permette l'eliminazione delle variabili globali che sono ormai ritenute una pratica obsoleta; eliminando l'utilizzo delle variabili globali avremo la possibilità di scrivere codice più ordinato, facilmente manutenibile e meno propenso agli errori. Il diagramma UML del design pattern Singleton è il seguente:

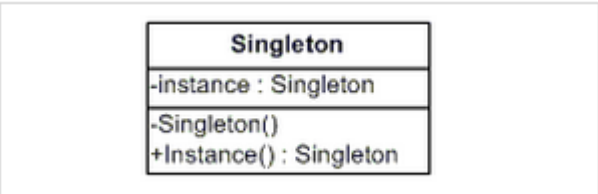

Figura 4.11: Design Pattern Singleton

Le funzioni implementate in questa classe sono:

- *ceckSession()*: gestisce i parametri di sessione per ogni client; più in particolare, per ogni utente autenticato nel sistema vengono salvate nei *cookie* le credenziali per consentirgli la navigazione tra le pagine; se così non fosse l'utente sarebbe costretto a effettuare il login per ogni pagina che visita. Inoltre viene impostato a 20 minuti il tempo massimo di inattività, dopo questo periodo di tempo il client viene disconnesso e deve rieseguire il *login* per poter continuare ad utilizzare il servizio. La funzione è la seguente:

```php
<?php
function checkSession() {
    session_start (); // create or restore session

    if (isset ( $_SESSION ['time'] ) && $_SESSION ['time'] < time () - 1200) {//session time:
        // logout                                                          20 min = 1200 seconds
        session_unset (); // empty session
        session_destroy (); // destroy session
                            // redirect to home page
        header ( 'Location:home.php?msg=SessionTimeOut' );
        return;
    }
    $res = array ();
    if (isset ( $_SESSION ['email'] ) && isset ( $_SESSION ['restaurantName'] )) {
        $res ['email'] = $_SESSION ['email']; // email_manager
        $res ['restaurantName'] = $_SESSION ['restaurantName'];
        $_SESSION ['time'] = time (); // update time
        return $res;
    } else {
        return $res;
    }
}
```

Figura 4.12: funzione PHP di gestione della sessione

- *sanitizeString($var)*: questa funzione ripulisce la stringa passata come argomento (lato server) da eventuali caratteri non validi attraverso l'uso di metodi PHP appositi:

```php
function sanitizeString($var) {
    $var = trim ( filter_var ( $var, FILTER_SANITIZE_STRING ) ); // strips or encodes unwanted characters
    $var = strip_tags ( $var ); // HTML and PHP tags removed
    $var = htmlentities ( $var ); // converts all special characters into HTML entities
    $var = stripslashes ( $var );
    return $var;
}
```

Figura 4.13: funzione sanitize String

- *setDB()*: è la funzione che crea la connessione con il database "*MySQL*" per consentire l'accesso ai dati. tramite la chiamata PHP "*mysqli_ connect*" che accetta i parametri "*db_ host*", "*db_ user*", "*db_ password*" e "*db_ name*":

```php
function setDB() {
    $dbhost = '127.0.0.1';
    $dbname = 'mybrmdb';
    $dbuser = 'root';
    $dbpass = '';
    $appname = "myBRM";

    $conn = @mysqli_connect ( $dbhost, $dbuser, $dbpass, $dbname );

    if (mysqli_connect_errno ()) {
        echo "Error connesion DBMS: " . mysqli_connect_error ();
        return;
    }
    return $conn;
}
```

Figura 4.14: funzione PHP per connessione al database

- *destroySession()*: utilizza la funzione PHP "*session_ destroy()*" che elimina tutti i dati associati alla sessione corrente. Non desetta nessuna delle variabili globali associate alla sessione o desetta il cookie di sessione.

- *nav_ bar()*: la barra di navigazione è implementata in questa classe in modo da non doverla ricreare per ogni pagina del sito:

```php
function nav_bar() {
    echo "<li><a href=\"home.php\">Home       </a></li><br>";
    echo "<li><a href=\"about.php\">About       </a></li><br>";
    echo "<li><a href=\"signup.php\">Sign Up     </a></li><br>";
    echo "<li><a href=\"login.php\">Log in      </a></li><br>";

    echo "<li><a href=\"insertreservation.php\">Insert Reservation</a></li><br>";
    echo "<li><a href=\"managereservation.php\">Manage Reservation</a></li><br>";
    echo "<li><a href=\"logout.php\">Log out       </a></li><br>";
}
```

Figura 4.15: funzione PHP barra di navigazione

4.2 Navigazione del sito

In questa sezione viene illustrato il funzionamento del prodotto finito "***My Restaurant Booking Manager***", con gli screenshot delle pagine e parti di codice più significative con brevi commenti.

4.2.1 Home Page

La *home page* per un utente che accede al sito la prima volta si presenta in questo modo:

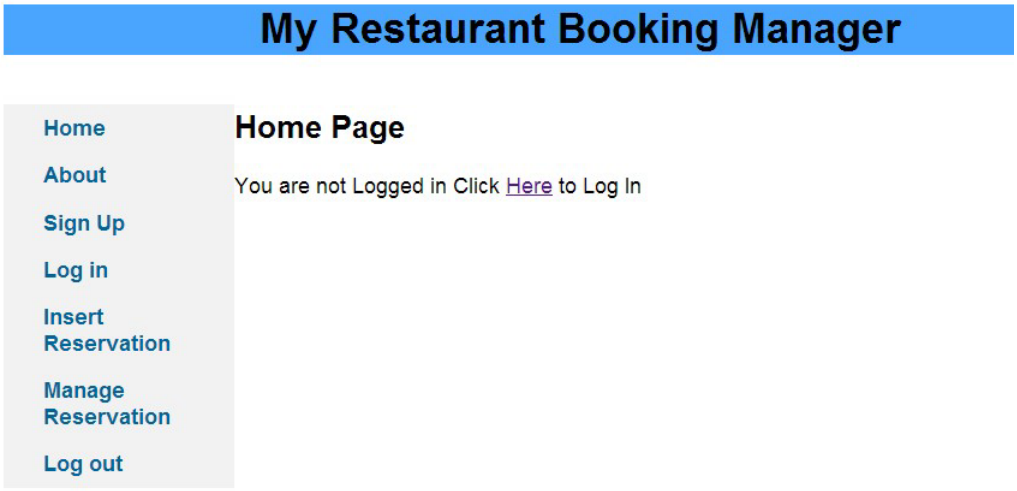

Figura 4.16: screenshot Home page utente non autenticato

L'utente cliccando sul link proposto viene reindirizzato alla pagina di *login*.

Se invece l'utente ha già effettuato l'accesso la *home* diventa:

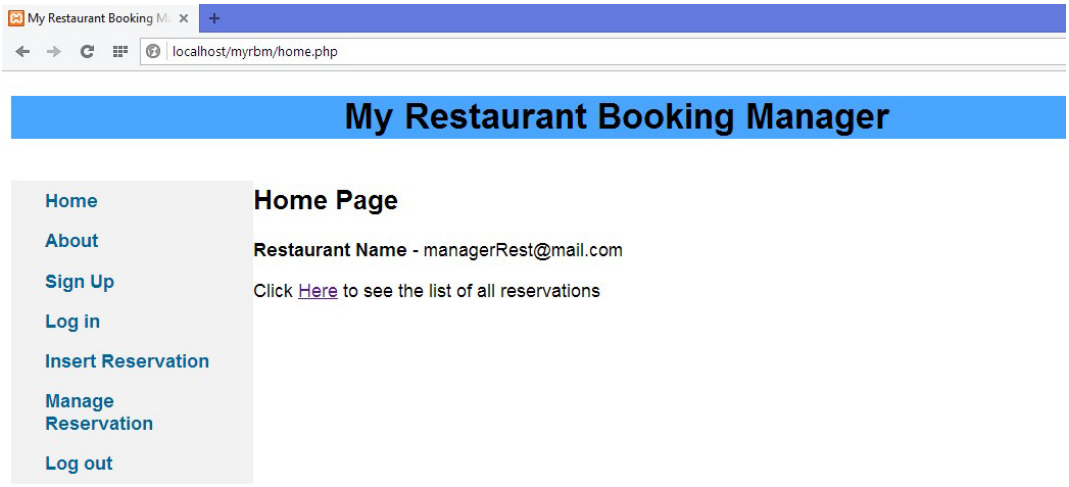

Figura 4.17: screenshot Home page

Questa è la pagina di benvenuto all'utente con il link per visualizzare la lista delle prenotazioni. Il link per visualizzare le prenotazioni fa un redirect alla sezione "*Manage Reservation*".

Il codice PHP che genera il contenuto da visualizzare nella pagina HTML è il seguente:

```php
<?php
include_once 'nb_functions.php';

$sess_arr = checkSession ();
$result = "";
$flag = 1;
if ($sess_arr) {
    $email = $sess_arr ['email'];
    $restaurantName = $sess_arr ['restaurantName'];

    $result = '<p><b>' . $restaurantName . '</b> - ' . $email . ' </p>';
    $result .= "<p>Click <a href=\"managereservation.php\"> Here</a> to see the list of all reservations </p>";
} else {
    if (isset ( $_GET ['msg'] ) && $_GET ['msg'] == "SessionTimeOut") {
        $result = "Session Expired ";
        $result .= '<p> Click <a href="login.php">Here</a> to Log In</p>';
    } else
        $result = '<p> You are not Logged in Click <a href="login.php">Here</a> to Log In</p>';
}
?>
```

Figura 4.18: codice php home page

Nel codice HTML è presente lo script che legge i *cookie* di sessione utilizzando la funzione "*ceckSession()*" della classe "*nb_function.php*" descritta precedentemente.

Lo stesso script è implementato in tutte le altre pagine lato server:

```
39      <noscript>This page needs JavaScript activated to work.</noscript>
40
41      <script type="text/javascript">
42          createCookie("test", "1");
43          var res = getCookie("test");
44          if(res!=1)
45              window.location.href = "index.php";
46          </script>
```

Figura 4.19: script che crea i cookies di sessione

la prima riga di codice viene visualizzata eventualmente se nel browser in uso non è abilitato Javascript.

Nella parte HTML il menu di navigazione visualizzato a sinistra viene generato, come in tutte le altre pagine, dalla funzione "nav_bar()" disponibile nella classe "nb_function.php":

```
57      <div id="navBar">
58          <ul id="nav">
59          <?php   nav_bar();  ?>
60          </ul>
61      </div>
62
```

Figura 4.20: codice HTML che visualizza il menu

4.2.2 About

La pagina *About* (Fig. 4.21) fornisce le caratteristiche della *Web application,* per consentire all'utente di conoscere le funzionalità del sito, e il contatto dell'autore mediante la pagina Google Plus.

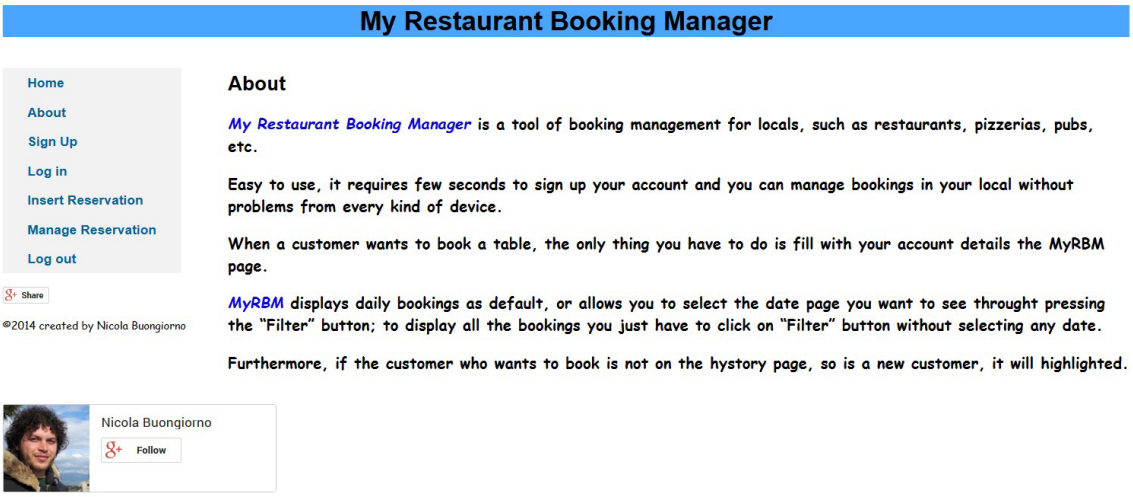

Figura 4.21: Pagina About

4.2.3 Sign Up

La pagina *Sign Up* consente la registrazione di un utente al sistema, memorizzando i suoi dati nel database. I campi da inserire sono:

- *Restaurant name*: il nome del ristorante che si vuole gestire
- *E-mail*: indirizzo mail del gestore
- *Password*: parola segreta per garantire la sicurezza dell'account.

La pagina HTML è visualizzata in questo modo:

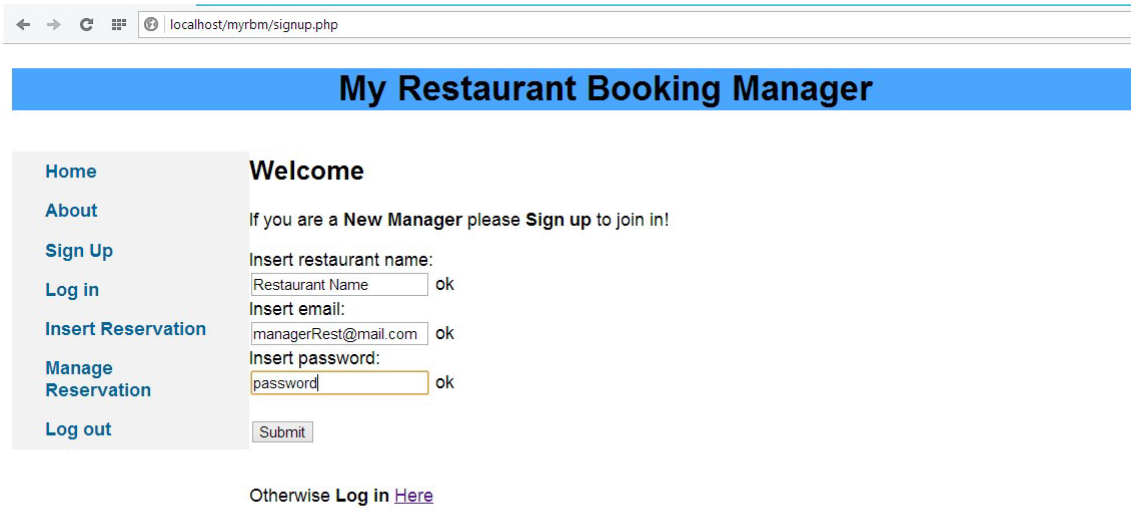

Figura 4.22: screenshot pagina Sign Up

Una volta che l'utente ha inserito i dati vengono salvati nel database per consentire successivamente la procedura di *login*.

Il form che consente l'inserimento dei dati effettua una chiamata *Post* su se stesso (con una chiamata di tipo *POST* si può richiedere una risposta al servente inviandogli contemporaneamente dei dati, da inserire nel corpo della richiesta, come argomento del metodo *send*) in questo modo:

```
132             If you are a <b>New Manager</b> please <b>Sign up</b> to join in!
133         </p>
134         <form method="post" action="<?php echo $_SERVER['PHP_SELF']; ?>">
135             Insert restaurant name: <br> <input id="restaurantName" type="text"
136                 name="textRestaurantName" value="<?php echo $restaurantName; ?>"
137                 onkeyup='validateString("restaurant")'> <span
138                 id="restaurantNameLabel"><?php echo $okRes; ?></span><br> Insert
139             email: <br> <input id="email" type="text" name="textEmail"
140                 onkeyup='validateEmail_Sign()'> <span id="emailLabel"></span><br>
141             Insert password: <br> <input id="passName" type="text"
142                 name="textPassName" onkeyup='validateString("pass")'> <span
143                 id="passNameLabel"></span><br> <br> <input id="submit"
144                 name="submit" type="submit" value="Submit" disabled>
145         </form>
```

Figura 4.23: metodo Post per inserimento dati

il controllo sulla correttezza dei campi viene eseguito con il seguente codice PHP:

```php
if (isset ( $_POST ['submit'] ) && (trim ( $_POST ['submit'] ) == "Submit")) {
    if (! isset ( $_POST ['textRestaurantName'] ) || $_POST ['textRestaurantName'] == "") {
        echo "Invalid Restaurant Name <br>";
        echo "<p><a href=\"home.php\">Come back to Sign up </a></p>";
        return;
    } elseif (! isset ( $_POST ['textEmail'] ) || $_POST ['textEmail'] == "" ||
            ! filter_var ( $_POST ['textEmail'], FILTER_VALIDATE_EMAIL )) {
        echo "Invalid Email <br>";
        echo "<p><a href=\"home.php\">Come back to Sign up </a></p>";
        return;
    } elseif (! isset ( $_POST ['textPassName'] ) || $_POST ['textPassName'] == "") {
        echo "Invalid Password <br>";
        echo "<p><a href=\"home.php\">Come back to Sign up </a></p>";
        return;
    } else {
        $restaurantName = sanitizeString ( $_POST ['textRestaurantName'] ); // Sanitize input
        $email = sanitizeString ( $_POST ['textEmail'] ); // Sanitize input
        $password = sanitizeString ( $_POST ['textPassName'] ); // Sanitize input
        $password = md5 ( $password );

        $conn = setDB ();
```

Figura 4.24: codice PHP controllo campi input

se i campi risultano corretti e non vuoti, vengono effettuati i controlli di tipo "*sanitize*" sui valori inseriti nei campi di input. Se i dati risultano corretti viene aperta la connessione con il database per aggiungere le informazioni dell'utente nella tabella "*account*" (*query* di inserimento dati), nel seguente modo:

```php
        $query = "INSERT INTO mybrmdb.account (email_manager, name_restaurant,  password) VALUES
                                    ('" . $email . "', '" . $restaurantName . "', '" . $password . "')";
        $res = @mysqli_query ( $conn, $query );

        if ($res) {
            // Registration Completed!
            $_SESSION ['email'] = $email;
            $_SESSION ['restaurantName'] = $restaurantName;
            mysqli_close ( $conn );

            header ( "Location: managereservation.php" );
            return;
        } else {
            echo "Registration failed ";
            echo "<p><a href=\"home.php\">Come back to Sign up </a></p>";
            return;
        }

```

Figura 4.25: query inserimento dati account nel database

se l'inserimento dati è andato a buon fine l'utente viene reindirizzato alla pagina "*manage reservation*", altrimenti è invitato a ripetere la procedura di registrazione.

4.2.4 Log in

La pagina di *Login* è molto semplice, richiede all'utente indirizzo e-mail di registrazione e password scelta. Se l'e-mail o la password risultano corrette il client viene reindirizzato alla pagina contenente la lista delle prenotazioni del giorno attuale, se invece e-mail o password (o entrambe) risultano errate o non presenti nel database, l'utente viene avvisato e invitato a riprovare il login (se l'utente non è registrato può selezionare la pagina "*Sign Up*").

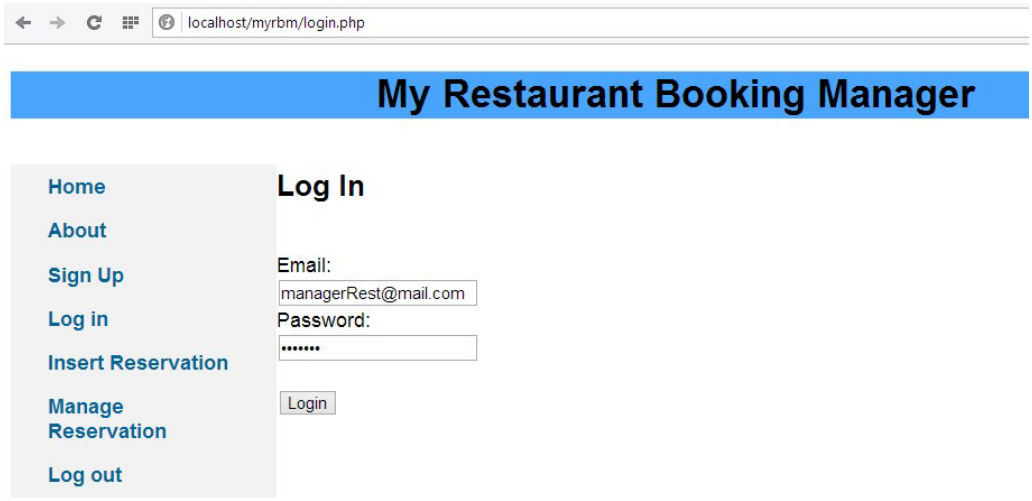

Figura 4.26: screenshot pagina Login

L'inserimento dei dati avviene sempre con una chiamata di tipo *POST* alla pagina stessa:

```
128             <form method="post" action="<?php echo $_SERVER['PHP_SELF']; ?>">
129                 Email: <br> <input id="managerEmail" type="text"
130                     name="textManagerEmail"> <br> Password: <br> <input type='password'
131                     name='password' /><br /> <br> <input id="submit" name="submit"
132                     type="submit" value="Login">
133             </form>
```

Figura 4.27: metodo Post per inserimento dati di autenticazione

Una volta effettuato il controllo della correttezza dei dati inseriti (simile al codice della pagina "*Sign Up*") viene fatta una *query* di selezione nella tabella

account per verificare i dati di autenticazione per un determinato utente; se la procedura di autenticazione è andata a buon fine il client viene reindirizzato alla pagina di gestione delle prenotazioni, altrimenti vine invitato a rieseguire il login:

```php
if (strpos ( $managerEmail, '@' ) !== false) {
    // TABLE ACCOUNT
    $query = "SELECT * FROM mybrmdb.account WHERE email_manager= '" . $managerEmail . "' and password = '" . $password . "'";

    $res = @mysqli_query ( $conn, $query );

    if ($res) {
        if (mysqli_num_rows ( $res ) > 0) {

            if (($row = mysqli_fetch_array ( $res )) != NULL) {
                $_SESSION ['email'] = $managerEmail;
                $_SESSION ['restaurantName'] = $row ["name_restaurant"];

                $result .= "<b>Restaurant Name:</b>" . $row ["name_restaurant"] . " <br><br>";
                $result .= "Now you are logged in!";
            } else
                $result .= "Invalid";

            mysqli_free_result ( $res );
        } else {
            $result = "Invalid credentials <br>";
            $result .= "<p><a href=\"login.php\">Come back to Log in</a></p>";
        }
    } else
        die ( "Operation failed " );

    mysqli_close ( $conn );

    header ( "Location: managereservation.php" );
    return;
}
```

Figura 4.28: query di selezione per autenticazione dell'utente

4.2.5 Insert Reservation

La pagina *Insert Reservation* svolge una delle funzioni principali, cioè consente l'inserimento della prenotazione di un tavolo. Per l'inserimento dei dati relativi al cliente che vuole prenotare il form è il seguente:

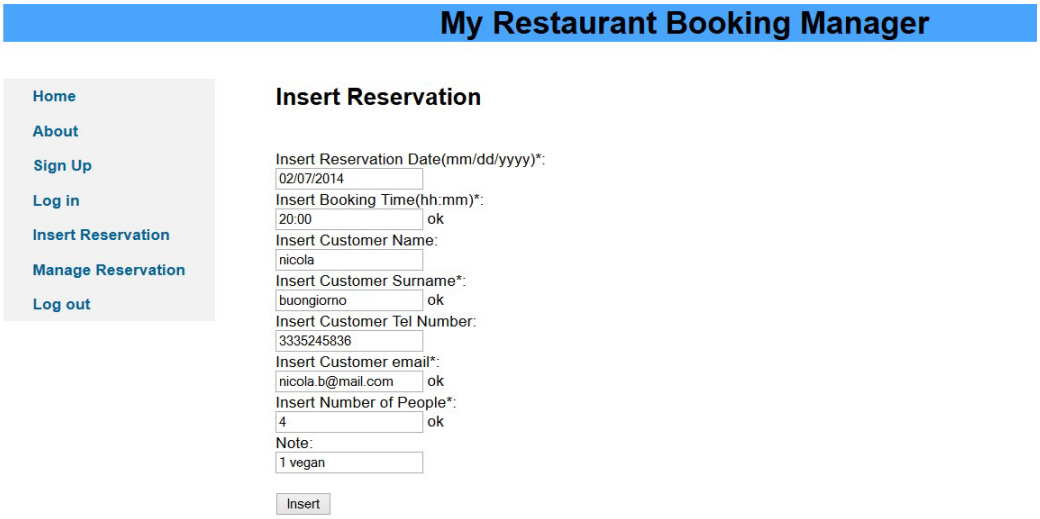

Figura 4.29: screenshot pagina Insert reservation

Il form *HTML* del metodo *POST* per l'inserimento dei dati è strutturato in questo modo:

```
216         <form method="post" action="<?php echo $_SERVER['PHP_SELF']; ?>">
217
218             Insert Reservation Date(mm/dd/yyyy)*: <br> <input id="date"
219                 type="text" name="date" value="<?php echo date("m/d/Y"); ?>"> <br>
220             Insert Booking Time(hh:mm)*: <br> <input id="AddBookingTime"
221                 type="text" name="textAddBookingTime"
222                 onkeyup='validateTime("AddBooking")'> <span
223                 id="AddBookingTimeLabel"></span><br> Insert Customer Name: <br> <input
224                 id="custAddName" type="text" name="textCustAddName"> <span
225                 id="custAddNameLabel"></span><br> Insert Customer Surname*: <br> <input
226                 id="custSurAddName" type="text" name="textCustSurAddName"
227                 onkeyup='validateName("custSurAdd")'> <span
228                 id="custSurAddNameLabel"></span><br> Insert Customer Tel Number: <br>
229             <input id="telNum" type="text" name="textCustTelAddName"> <span
230                 id="telNumLabel"></span><br> Insert Customer email*: <br> <input
231                 id="email" type="text" name="textEmailC" onkeyup='validateEmail()'>
232             <span id="emailLabel"></span><br> Insert Number of People*: <br> <input
233                                 id="peopleNum" type="text" name="peopleNum"
234                 onkeyup='validateNum("people")'> <span id="peopleNumLabel"></span><br>
235             Note: <br> <input id="note" type="text" name="note"> <span
236                     id="noteLabel"></span><br> <br> <input id="submit" name="submit"
237                     type="submit" value="Insert" disabled>
238         </form>
```

Figura 4.30: form Html inserimento dati

Per ogni campo di inserimento viene effettuato il controllo con il relativo metodo dello script "*validate.js*" per stabilire la correttezza delle informazioni inserite al

fine di abilitare o meno il tasto "*submit*" (il tasto "submit" viene abilitato solo se tutti i campi obbligatori sono seguiti dalla dicitura "*ok*" in *HTML* che è il risultato dell'elaborazione dei metodi presenti nello script "*validate.js*"). Omette il codice relativo a suddetti controlli in questo documento perchè è molto simile alla procedura di registrazione dell'utente. La query di inserimento della prenotazione nella tabella "*reservations*", se la procedura è andata a buon fine, è la seguente:

```
124             $query = "INSERT INTO mybrmdb.reservation(id_reservation, email_manager, email_customer, name_c,
125                                                 surname_c, tel_c, time_booking, date,number_of_people,note,new_c) VALUES
126         ('" . $id_reservation . "', '" . $email_manager . "', '" . $textEmailC . "', '" . $textCustAddName . "',
127                     '" . $textCustSurAddName . "', '" . $textCustTelAddName . "', '" . $textAddBookingTimeSec . "',
128                             '" . $date . "', '" . $peopleNum . "', '" . $note . "', '" . $new_c . "')";
129             $res = @mysqli_query ( $conn, $query );
130 
131             if ($res) {
132                 // Insert Completed!
133                 $result = "Insert Completed! <br>";
134                 $result .= "<p>Click <a href=\"managereservation.php\"> Here</a> to see the list of all available reservations </p>";
135             } else {
136                 $result = "Insert Failed! <br>";
137                 $result .= "<p><a href=\"home.php\">Come back</a></p>";
138             }
```

Figura 4.31: query inserimento dati prenotazione

I campi obbligatori per effettuare una prenotazione sono contrassegnati da un asterisco "*", è possibile effettuare la prenotazione anche omettendo i dati non obbligatori come illustra il seguente esempio:

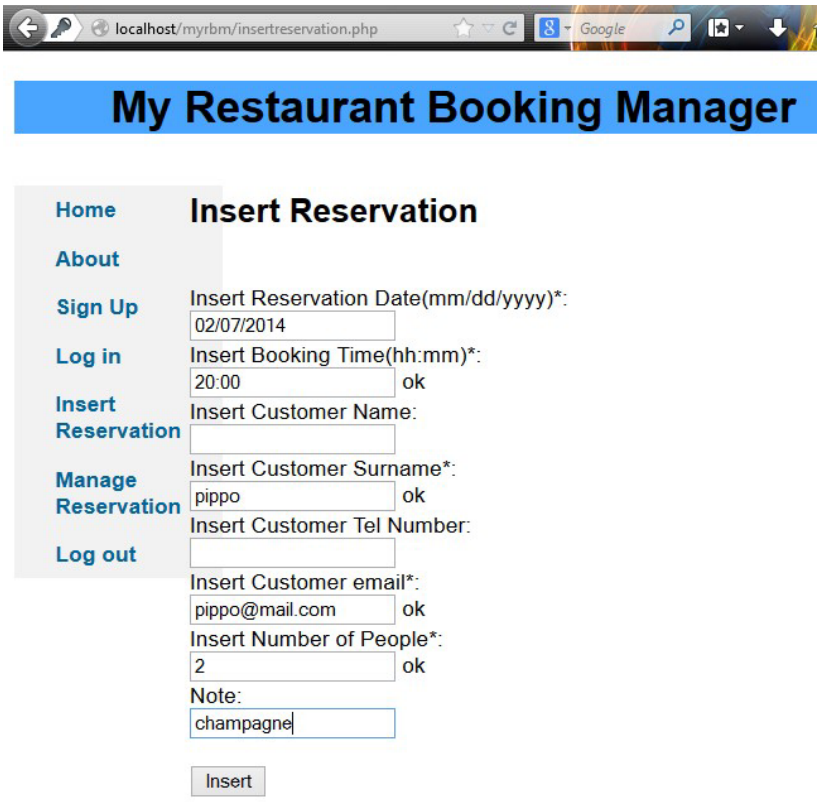

Figura 4.32: screenshot inserimento prenotazione

Per quanto riguarda il campo "New Customer" che indica se il cliente che ha prenotato è un nuovo cliente dell'attività ristorativa (cliente è diverso da client/utente, invece si riferisce all'agente esterno al sistema che richiede la prenotazione di un tavolo), viene effettuata una query di selezione per stabilire se per il manager in questione esiste già una prenotazione associata all'indirizzo e-mail del cliente:

```php
// check if is a New Customer
$query = "SELECT * FROM mybrmdb.reservation WHERE email_customer= '" . $textEmailC . "' AND email_manager= '" . $email_manager . "' ";

$res = @mysqli_query ( $conn, $query );

    if ($res) {
        if (mysqli_num_rows ( $res ) > 0) {
            // Old Customer
            $new_c = "";
            // Update the new customer status of the previous customer reservation
                        $sql = "UPDATE mybrmdb.reservation SET  new_c='" . $new_c . "'
                    WHERE email_customer= '" . $textEmailC . "' AND email_manager= '" . $email_manager . "' ";
            $res2 = @mysqli_query ( $conn, $sql );
            if (! $res2) {
                echo "Update failed for customr: " . $textEmailC . "";
            }
        } else {
            // New Customer
            $new_c = "Yes";
        }
    } else
        die ( "Operation failed while checking new customer" );
```

Figura 4.33: query di selezione nuovo cliente

se il cliente è nuovo viene visualizzato nella lista delle prenotazioni la voce "*Yes*" nel campo "*New Customer*"; se invece il cliente ha gia effettuato una prenotazione, significa che non è più nuovo cliente, quindi viene aggiornato il database rimuovendo la dicitura "*Yes*" impostata precedentemente per quel determinato cliente identificato dalla chiave "*email_ customer*".

L'inserimento della data viene effettuato tramite la funzione "*datapicker*" di *jQuery* per comodità e velocità di utilizzo:

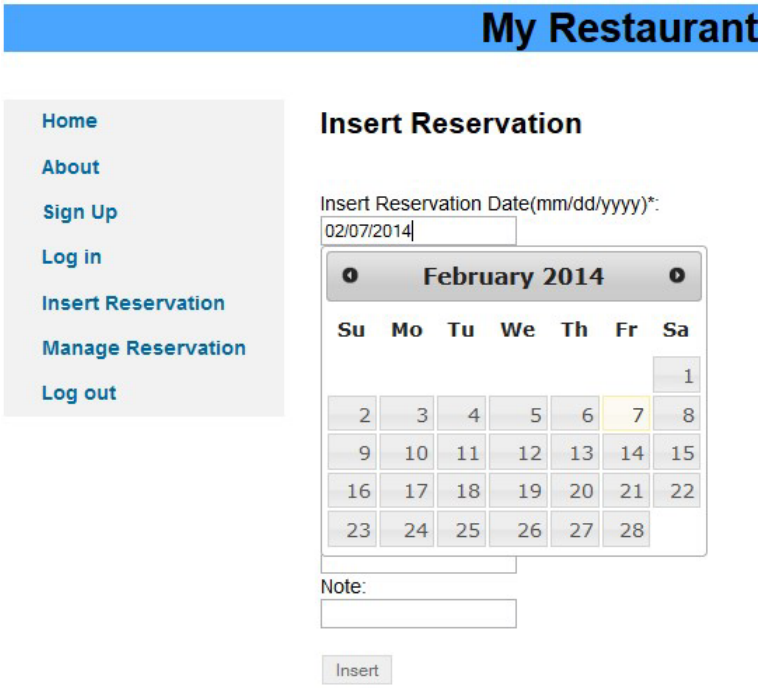

Figura 4.34: screenshot datapicker

il codice seguente richiama le librerie *jQuery* per utilizzare la funzione "*datapicker()*" per l'inserimento della data:

```
159     href="http://code.jquery.com/ui/1.10.3/themes/smoothness/jquery-ui.css" />
160 <script src="http://code.jquery.com/jquery-1.9.1.js"></script>
161 <script src="http://code.jquery.com/ui/1.10.3/jquery-ui.js"></script>
162
163 <script>
164   $(function() {
165     $( "#date" ).datepicker();
166   });
167   </script>
```

Figura 4.35: script jQuery datapicker

4.2.6 Manage reservation

Manage reservation è la pagina di visualizzazione e gestione delle prenotazioni. Di default questa pagina mostra la lista di prenotazione del giorno attuale:

Figura 4.36: screenshot pagina Manage Reservation con prenotazioni

Se per il giorno attuale non sono presenti prenotazioni la pagina è visualizzata in questo modo:

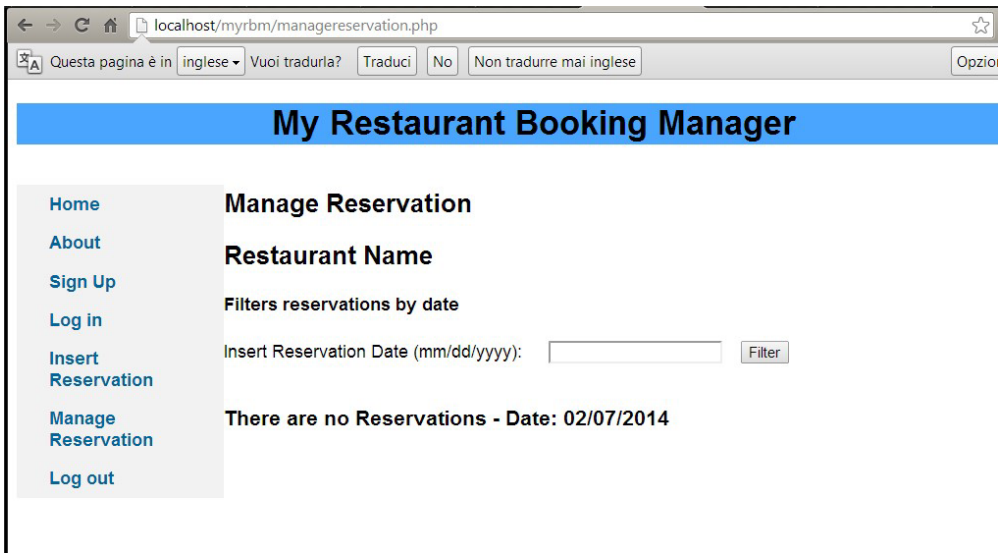

Figura 4.37: screenshot pagina Manage Reservation senza prenotazioni

Il tasto "*Filter*" consente di visualizzare le prenotazioni relative ad una data scelta dall'utente tramite la funzione *"datapicker"* della libreria *jQuery* allo stesso modo della procedura di inserimento della prenotazione:

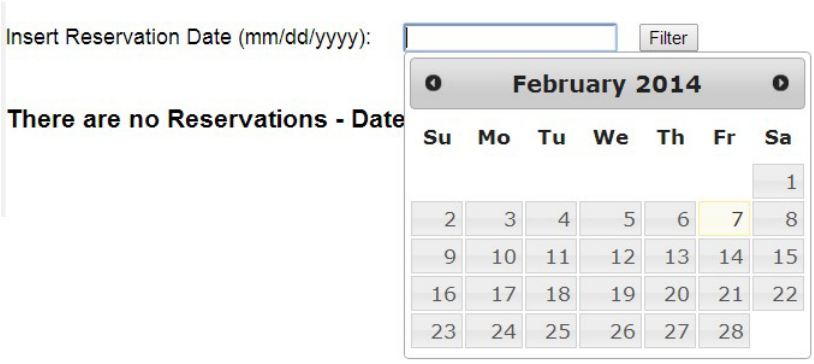

Figura 4.38: screenshot tasto Filter

Il form *HTML* che riceve la data in ingresso è una chiamata di tipo *GET* alla pagina stessa:

```
170     <form method="get" action="<?php echo $_SERVER['PHP_SELF']; ?>">
171             <h2> <?php echo $res_name ?> </h2>
172
173             <h4>Filters reservations by date</h4>
174             Insert Reservation Date (mm/dd/yyyy):     <input id="date"
175                 type="text" name="date">   <input id="submit" name="submit"
176                 type="submit" value="Filter">
177         </form>
```

Figura 4.39: metodo Html Get per filtrare la data

Se viene premuto il tasto "*Filter*" senza selezionare alcuna data, viene mostrata la lista di prenotazione di tutte le date (All Date):

Figura 4.40: screenshot Manage Reservation con tutte le prenotazioni

Il codice PHP che visualizza la tabella delle prenotazioni in ordine di data e ora, dopo aver controllato la correttezza dell'input (data da filtrare) è il seguente:

```php
if ($date == "null") {
    $query = "SELECT * FROM mybrmdb.reservation WHERE date= '"
        . date ( "m/d/Y" ) . "' AND email_manager= '" . $email_manager . "'  ORDER BY date, time_booking";
    $d = date ( "m/d/Y" );
} else if ($date == "all") {
    $query = "SELECT * FROM mybrmdb.reservation WHERE email_manager= '"
                        . $email_manager . "'  ORDER BY date, time_booking";
    $d = "All Dates";
} else {
    $query = "SELECT * FROM mybrmdb.reservation WHERE date= '"
                    . $date . "' AND email_manager= '" . $email_manager . "' ORDER BY date, time_booking";
    $d = $date;
}

$res = @mysqli_query ( $conn, $query );

if ($res) {
    if (mysqli_num_rows ( $res ) > 0) {

        $result = "<h3> Reservations - Date: " . $d . "</h3>";

        $result .= "<table border=1>";

        $result .= "<tr><th> Booking Time</th> <th> Date</th> <th>Surname</th> <th>Name</th>
              <th> Num of People</th> <th>Email</th> <th> Tel </th>
            <th> Note</th> <th> New Customer</th> <th>  Action   </th> <th>  Action   </th> </tr>";

        while ( ($row = mysqli_fetch_array ( $res )) != NULL ) {

            $result .= "<tr> ";
            $result .= "<td> <b>" . gmdate ( "H:i", $row ["time_booking"] ) . " </b></td><td> "
                    . $row ["date"] . '</td><td>' . $row ["surname_c"] . "</td><td>" . $row ["name_c"] . "</td><td>"
                            . $row ["number_of_people"] . "</td><td> " . $row ["email_customer"] . "</td><td>"
                                    . $row ["tel_c"] . "</td><td>" . $row ["note"] . '</td><td> <font color="red">'
                                            . $row ["new_c"] . " </font> </td>";
            $result .= '<td><form action="modifyreservation.php" method="post">
                                <input type="hidden" name="id_reservation" value="' . $row ['id_reservation'] . '" />
                                <input  id="submit" name="submit" type="submit" value="Modify" />
                                </form> </td> <td><form action="managereservation.php" method="post">
                                <input type="hidden" name="id_reservation" value="' . $row ['id_reservation'] . '" />
                                <input  id="submit" name="submit" type="submit" value="Cancel" />
                                </form></td>';

            $result .= "</tr>";
        }

        $result .= "</table>";
    } else {
        $result = "<h3>There are no Reservations - Date: " . date ( "m/d/Y" ) . "</h3>";
    }
```

Figura 4.41: codice PHP visualizzazione prenotazioni

i costrutti "*if-than-else*" controllano la data inserita che può essere "*null*" se non viene filtrata la data, quindi vengono visualizzate le prenotazioni del giorno attuale, "*all*" se viene premuto il tasto *filter* senza selezionare alcuna data, *la web-app* visualizza tutte le prenotazioni; oppure se viene selezionata una data specifica, vengono mostrate le prenotazioni di quella data se presenti.

Manage Reservation consente, inoltre, di effettuare modifiche sulle singole prenotazioni tramite il tasto "*Modify*" di fianco ad ogni prenotazione, oppure eliminare le singole prenotazioni mediante il tasto "*Cancel*". Se viene premuto il tasto "*Cancel*" la prenotazione viene eliminata direttamente dal database nel seguente modo:

```
36      $id_reservation = sanitizeString ( $_POST ['id_reservation'] ); // Sanitize input
37
38      $query = "DELETE FROM mybrmdb.reservation  WHERE id_reservation= '" . $id_reservation . "' AND email_manager= '" . $email_manager . "' ";
39          $res = @mysqli_query ( $conn, $query );
40          if (! $res) {
41              $result = "Reservation - Remove failed <br> " . mysqli_error ( $conn );
42              $result .= "<p><a href=\"managereservation.php\">Come back </a></p>";
43              echo "<div id=\"result\">";
44              echo $result;
45              echo "</div>";
46              return;
47          }
```

Figura 4.42: metodo PHP per la cancellazione di una prenotazione

se invece viene premuto il tasto "*Modify*" l'applicazione rimanda l'utente alla pagina "*Modify Reservation*".

4.2.6.1 Modify Reservation

La pagina "*Modify Reservation*" è simile a "*Insert Reservation*", i campi sono precompilati con le informazioni della prenotazione salvata nel database. E' possibile effettuare delle modifiche alla prenotazione e dopo i soliti controlli all'input premendo il tasto "*Update*" si aggiornano le informazioni presenti nel database. La visualizzazione *HTML* è la seguente:

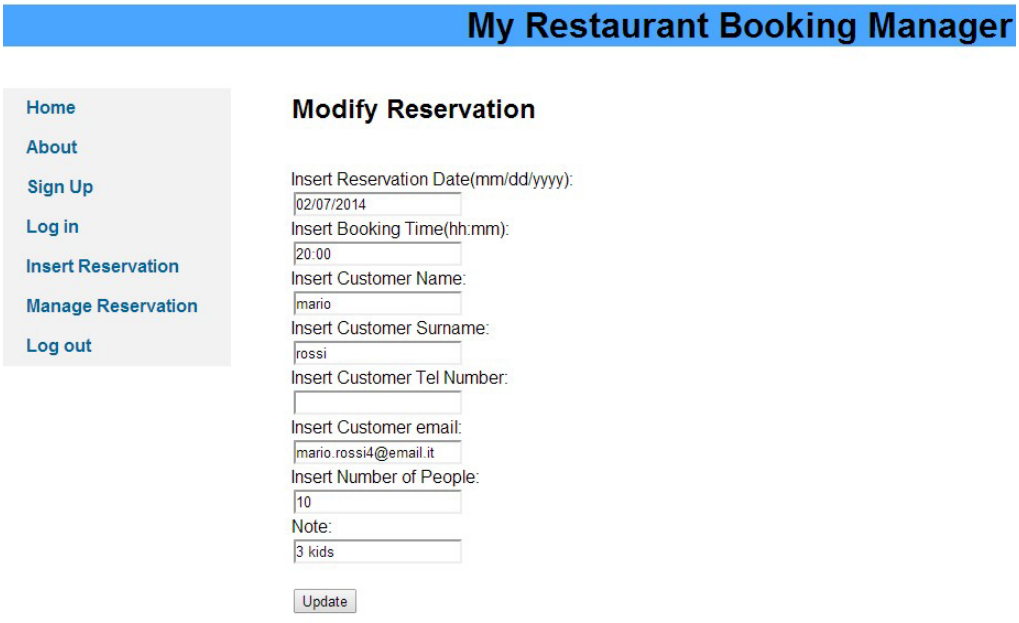

Figura 4.43: screenshot pagina Modify Reservation

Il form HTML che raccogli i dati di input, fornendo i campi precompilati dalla prenotazione effettuata in precedenza è:

```
259         <form method="post" action="<?php echo $_SERVER['PHP_SELF']; ?>">
260
261             Insert Reservation Date(mm/dd/yyyy): <br> <input id="date"
262                 type="text" name="date" value="<?php echo $varDate; ?>"> <br>
263                     Insert Booking Time(hh:mm): <br> <input id="AddBookingTime"
264                 type="text" name="textAddBookingTime"
265                     value="<?php echo $varTime; ?>"> <span id="AddBookingTimeLabel"></span><br>
266             Insert Customer Name: <br> <input id="custAddName" type="text"
267                 name="textCustAddName" value="<?php echo $varName; ?>"> <span
268                 id="custAddNameLabel"></span><br> Insert Customer Surname: <br> <input
269                 id="custSurAddName" type="text" name="textCustSurAddName"
270                 value="<?php echo $varSurname; ?>"> <span id="custSurAddNameLabel"></span><br>
271             Insert Customer Tel Number: <br> <input id="telNum" type="text"
272                 name="textCustTelAddName" value="<?php echo $varTel; ?>"> <span
273                 id="telNumLabel"></span><br> Insert Customer email: <br> <input
274                 id="email" type="text" name="textEmailC"
275                 value="<?php echo $varEmail; ?>"> <span id="emailLabel"></span><br>
276             Insert Number of People: <br> <input id="peopleNum" type="text"
277                 name="peopleNum" value="<?php echo $varNum; ?>"> <span
278                 id="peopleNumLabel"></span><br> Note: <br> <input id="note"
279                 type="text" name="note" value="<?php echo $varNote; ?>"> <span
280                 id="noteLabel"></span><br> <br> <input type="hidden"
281                 name="id_reservation" value="<?php echo $varId; ?>"> <input
282                 id="submit" name="submit" type="submit" value="Update">
283         </form>
```

Figura 4.44: metodo Post dati input per modifica prenotazione

Il codice che aggiorna il database dopo aver effettuato i controlli sugli input è il seguente:

```
 97             $sql = "UPDATE mybrmdb.reservation SET
 98             email_customer='" . $textEmailC . "',
 99             name_c='" . $textCustAddName . "',
100             surname_c='" . $textCustSurAddName . "',
101             tel_c='" . $textCustTelAddName . "',
102             time_booking='" . $textAddBookingTimeSec . "',
103             date='" . $date . "',
104             number_of_people='" . $peopleNum . "',
105             note='" . $note . "'
106             WHERE id_reservation='" . $idRes . "'";
```

Figura 4.45: codice che aggiorna una prenotazione nel database

4.2.7 Log out

La funzione *Logout* consente all'utente di scollegarsi dal sito cliccando "*Log out*" nella barra di navigazione a sinistra:

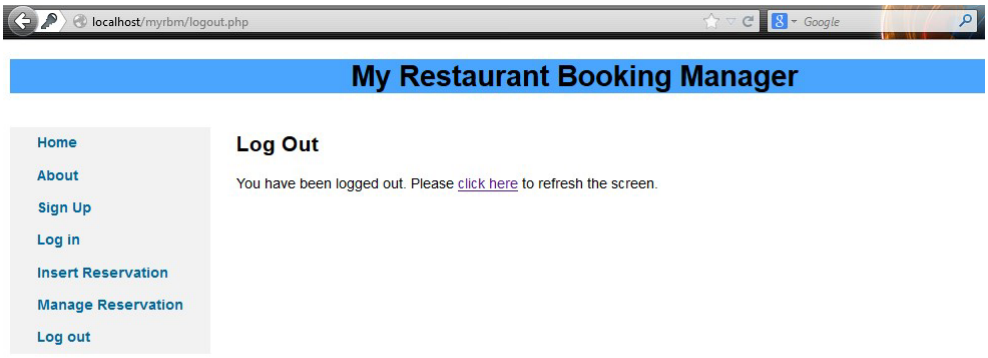

Figura 4.46: screenshot pagina di log out

Il codice *PHP* che esegue il *logout* e mostra il relativo messaggio a video è il seguente:

```
1  <?php
2  session_start ();
3  include_once 'nb_functions.php';
4
5  if (isset ( $_SESSION ['email'] ) || isset ( $_SESSION ['restaurantName'] )) {
6      session_unset ();
7      destroySession ();
8      $mess = "You have been logged out. Please <a href='home.php'>click here</a> to refresh the screen.";
9  } else
10     $mess = "You are not logged in";
11
12 ?>
```

Figura 4.47: codice PHP logout

4.3 Test

Il testing rappresenta una delle attività più importanti per assicurare la qualità del software. Dopo aver generato il codice sorgente, si deve collaudare il software per scoprire (e correggere) quanti più errori possibili prima di rilasciare il prodotto. Quando il software è stato realizzato, spesso è necessario apportare delle modifiche, sia perché il software non risulta rispettare completamente le specifiche fissate, sia perché si ritiene necessario perfezionare alcune funzionalità e caratteristiche. Si

possono fare piccole modifiche oppure, se siamo molto lontani dai requisiti, si può scegliere di fare la reingegnerizzazione del sistema, in pratica ripartire da capo. Finita con successo la fase di manutenzione si può eseguire il rilascio del sistema. La parte di reingegnerizzazione del sistema che effettuando i test è avventua una volta non viene riportata in questo documento.

I test sono stati condotti simulando tanti possibili scenari d'uso propri e impropri al fine di testarne la robustezza funzionale; sono stati condotti quindi dei test di utilizzo della *webapp* su più dispositivi, anche contemporaneamente, e sono stati simulati quasi tutti i possibili scenari di utilizzo.

Gli aspetti dell'applicazione su cui sono stati effettuati i test con esito positivo riguardano principalmente:

- Database: sono stati effettuati tutti i test di inserimento, interrogazione e cancellazione sul database (*query*) per verificarne il corretto funzionamento;
- Compatibilità: utilizzo della web-app in browser differenti e dispositivi differenti; per la simulazione di una grande varietà di dispositivi è stato utilizzato il servizio web "*responsivepx*"[1] che consente di simulare il layout della web app sui diversi dispositivi (smartphone e tablet) che differiscono per le dimensioni dello schermo; [TRD]
- Sicurezza: inserimento di dati validi nei rispettivi campi di input, gestione in sicurezza dei dati memorizzati nel database;
- Concorrenza: lo stesso account (username e password) può essere utilizzato da più utenti contemporaneamente; il database è utilizzato in concorrenza anche quando diversi account interagiscono con la web app allo stesso momento.

L'unico test fallito è rappresentato in figura 4.47 e riguarda l'inserimento delle informazioni nei campi di input;

se l'inserimento dell'informazione (compilazione campo input) avviene mediante digitazione dalla tastiera o "copia e incolla", viene effettuato il normale controllo dell'input e in caso di esito positivo si abilita il pulsante di inserimento;

[1] Il sito web del servizio è: *http://responsivepx.com*

se invece l'inserimento dell'input avviene mediante selezione di un informazione inserita precedentemente dal browser (che il browser ricorda) non viene eseguito il controllo sui campi di input[2], quindi è uguale ad un campo vuoto (il tasto di inserimento rimane disabilitato) ed è necessario digitare almeno un carattere nel campo di input per l'esecuzione della procedura di controllo. Dalla figura 4.47 si nota che non viene eseguito il controllo di input, infatti di fianco ai campi non è visualizzato (in HTML) il risultato del controllo ("ok" oppure "invalid").

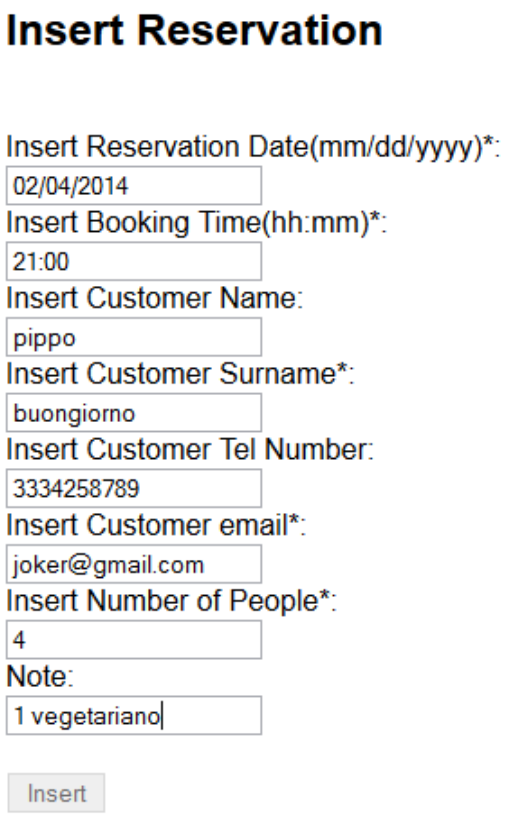

Figura 4.48: Inserimento dati di prenotazione mediante selezione dell'input dal browser

[2]Il controllo dei campi di input è la funzione che se risulta positiva visualizza nel browser "ok" di fianco al campo di inserimento, altrimenti visualizza "invalid" e non viene abilitato il tasto di inserimento.

Capitolo 5

Conclusioni

In conclusione, questa tesi presenta una soluzione efficiente per la gestione delle prenotazioni di un'attività ristorativa (es. pizzeria, ristorante, trattoria, bar ecc.). I risultati ottenuti rispettano i vincoli di progetto discussi in fase di analisi. L'applicazione web creata (*My Restaurant Booking Manager*) permette la gestione delle prenotazione a tutti i ristoratori (manager) che intendono usarla; infatti hanno un accesso completo ai loro dati (di prenotazione) contemporaneamente; infatti il sistema gestisce oltre che la concorrenza di più account che usano la Web app allo stesso momento, anche la concorrenza di più utilizzatori dello stesso account, molto utile per la condivisione del lavoro tra più collaboratori.

Il sito *MyRBM* è attualmente offline in attesa di perfezionamenti della grafica e la creazione del logo.

5.1 Stima del costo

Si è cercato a questo punto di fare una stima del costo per produrre la *web application "MyRBM"* mediante il modello ***COCOMO***, che è il metodo più diffuso di stima dei costi del software.

Il *COCOMO* [Coc11] è un modello matematico usato da chi progetta software per stimare alcuni parametri fondamentali come il tempo di consegna e i mesi-persona necessari per lo sviluppo. Con questo metodo si cerca di fare una stima dello sforzo impiegato per la realizzazione di un progetto software.

La formula usata è:

$$Performance = (Complexity)\ \hat{}\ (Process) * (Team) * (Tools)$$

- *Performance* = Sforzo oppure tempo
- *Complexity* = Volume del codice generato
- *Process* = Maturità e metodo
- *Team* = Abilità, esperienza, motivazione
- *Tools* = Automazione del processo

Considerando la complessità del mio progetto possiamo applicare lo standard dei parametri applicati ai progetti di tipo Base, cioè quelli identicati come *"Organic mode"*[1].

Avremo quindi:

$$Sforzo = 2{,}4 * (\ 1{,}3\ \hat{}\ 1{,}05\ ) * 1 = \mathbf{3{,}16\ mp}$$

5.2 Sviluppi futuri

In questo momento gli utenti che possono utilizzare *MyRBM* sono esclusivamente gestori di attività ristorative, in futuro mi piacerebbe sviluppare la parte "*client*" della *web app*, ovvero il servizio che ogni gestore di attività ristorative potrà offrire ai propri clienti; aggiungendo quindi una funzionalità molto utile che fa risparmiare tempo al gestore. Più in particolare la parte da sviluppare riguarda l'aggiunta di una pagina web da cui i clienti del gestore dell'attività ristorativa possano prenotare. Il link della pagina web che consente ai nuovi utenti di prenotare viene reso pubblico dal gestore sottoforma ad esempio di *QR Code (Quick Response code)* o link, nel sito dell'attività ristorativa oppure in formato cartaceo (QR Code).

[1] URL del sito di riferimento: *http://www.softstarsystems.com/overview.htm*

Bibliografia

[Fab03] Fabris G.P., *Il nuovo consumatore: verso il postmoderno*, F.Angeli, Milano, 2003

[DinFab05] Di Nallo E., Fabris G.P. (a cura di), *L'esperienza del tempo di consumo tra pratiche e fruizione sociale*, F. Angeli, Milano, 2005

[PreRog07] Pressman, Roger S., *Principi di ingegneria del software*. Eds. Maura Cerioli, and Gianna Reggio. McGraw-Hill, 2007

[Con03] Conalen J., *Applicazioni Web con UML*, 2° edizione italiana, Pearson Education, Italia, 2003

[ArlNeu06] Arlow J., Neustadt I., *Uml 2 e unified process*, 2° ed., McGraw-Hill, 2006

[Eck11] Eckerson W. W., *Three tier Client/Server architectures: Achieving scalability, performance, and efficiency in Client/Server applications.* Open Information Systems, 1995

[ACPT09] Atzeni P., Ceri S., Paraboschi S., Torlone R., *Basi di dati, modelli e linguaggi di interrogazione*, 3° ed. McGraw-Hill, 2009

[StoVas04] Stobart S., Vassileiou M., *PHP e MySQL. Guida completa*, Apogeo, 2004

[GreBul01] Greenspan J., Bulger B., *Sviluppare applicazioni per database con MySQL/PHP* . Coll. ApogeoPro 2001

[Wan06] Wandschneider M., *Sviluppare applicazioni web con PHP e MySQL*, Apogeo, 2006

[AvvCim11] Avvenuti M., Cimino M.G.C.A., *Laboratori di programmazione web*, McGraw-Hill, Febbraio 2011 - a Pisa c/o le Librerie CLU, Pellegrini e LTU

[Tan02] Tansley D., *Pagine web dinamiche con PHP e MySQL*, Pearson, 2002

[Coc11] *Modello COCOMO*, http://csse.usc.edu/csse/research/COCOMOII/cocomo_main.html, http://www.softstarsystems.com/overview.htm

[Lam08] Lamanna C.: *Guida CSS di base.* CSShtml.it [Online] http://css.html.it/guide/leggi/2/guida-css-di-base/

[W3S] *W3Schools Online Web Tutorials*, http//:www.w3schools.com, 20/01/2014

[PMA13] *PhpMyAdmin*, http//:www.phpmyadmin.net/home_page/index.php, 01/02/2014

[Apa] *Apache web server*, http://www.apachefriends.org/it/index.html, 25/01/2014

[Jqu] *jQuery*, http//:www.jquery.com, 02/12/2013

[TRD] *Tool for Responsive Design*, http://responsivepx.com

[Xam] *XAMPP*, http://www.apachefriends.org/it/index.html

[Luc] *Lucid chart*, https://www.lucidchart.com