immobilUnisa

UNIVERSITÀ DEGLI STUDI DI SALERNO

Facoltà di Scienze Matematiche Fisiche e Naturali

Dipartimento di Informatica

Ingegneria del Software

Anno Accademico 2015/16

Progetto: immobilUnisa Versione 1.1
18/12/2015

Object Design Document

Componenti Del Gruppo

Giuseppe Di Martino	0512101162
Luca Diodato De Martino	0512102658
Giuseppe Giordano	0512101798
Francesco Napoli	0512101928

Sommario

RevisionHistory

Data	Versione	Descrizione	Autore
16/12/2015	1.0	Specifica delle interfacce dei moduli da implementare	Tutto il gruppo
18/12/2015	1.1	Versione completa	Tutto il gruppo

1. Introduction

1.1 Object design trade-offs

In questo documento vengono descritte le funzionalità del sistema e verranno introdotte tutte le caratteristiche scelte in base alla loro applicazione n quest'ultimo, si tenga presente che le funzionalità sviluppate non completano il sistema ma sono state scelte da implementare solo quelle funzionalità che si ritenevano cruciali e di vitale importanza all'esistenza del sistema stesso.

Le funzionalità che faranno parte dell'implementazione saranno:

1)La parte relativa all'agente

2)La parte relativa al cliente

3)La parte dell'amministratore

Escludendo di conseguenza la funzionalità di messaggistica interna del sistema tra le varie utenze e i settori del sistema ritenuti di importanza minore come la scheda contatti. Abbiamo scelto di sviluppare il sistema sfruttando le potenzialità del linguaggio HTML per quanto riguarda le parti statiche delle pagine lato client, è stato invecesfruttato il linguaggio PHP per tutte le funzioni dinamiche lato server.

Per quanto riguarda la parte di memorizzazione delle informazioni ci siamo serviti del software MySQL, un DBMS open-source e facilmente adattabile a tutti i server attualmente utilizzati.

Interfaccia vs usabilità
L'interfaccia è state realizzata sfruttando elementi di facile comprensione dove un utente medio dovrebbe riuscire a intuire le azioni che compie, inoltre ad ogni elemento è associato un testo leggibile che ne esplica la funzione.

Comprensibilità vs tempo
Per rendere comprensibile il codice anche a chi non lo ha implementato direttamente, sono state aggiunti molti commenti alle linee di codice per renderne immediata la comprensione, questo ha richiesto un lasso di tempo pari ad un quinto del tempo impiegato a scrivere il resto del codice.

Sicurezza vs efficienza
La sicurezza è uno degli aspetti più importanti per quanto riguarda un sistema client-server considerata la moltitudine e la varietà di attacchi che gli possono essere portati. In sistemi di questo tipo, si scrivono algoritmi complessi per codificare e decodificare le comunicazioni tra le parti in modo che anche se terzi riuscissero ad appropriarsi dei pacchetti di dati, non possano facilmente decodificarli e reperire informazioni private. Purtroppo il tempo a nostra disposizione non è sufficiente per poter implementare tali sistemi di sicurezza.
Si è optato per implementare una sicurezza che si basa su una combinazione di utente e password per interagire col sistema. Tali controlli sono anche efficienti avendo un ritardo praticamente nullo.

Tempo di rilascio vs Tolleranza ai fault
Nel nostro sistema non sono stati previsti meccanismi di recupero di eventuali dati dispersi per mancanza dei tempi di rilascio stabiliti.

1.2. Interface DocumentationGuidelines
Le classi PHP sono state strutturate in modo che ognuna rappresenti un'entità da manipolare, implementata tramite un solo file che ne descrive chiaramente le funzionalità interne. Inoltre ad ogni entità primaria è stata associata una façade ed una interfaccia. La classe pricipale sarà quella del database comprendente tutte le istruzioni elementari per leggere e scrivere dati.

Ogni file sorgente PHP è strutturato nel seguente modo:
-Istruzione/iInclude_once:

Serve per includere oggetti dichiarati in altri file PHP

-Descrizione della classe:

Una descrizione che comprende anche informazioni sulla versione e sulla data di creazione

-Dichiarazione di classe:

Formata da codice implementato, che comprende: variabili globali, variabili di istanza, funzioni, costruttori ed eventuali commenti locali.

Si fa notare che ogni volta che si dovrà aprire un file di codicecon un software di programmazioneutilizzeremonotepad++, il codice viene colorato per renderlo più leggibile, identificare meglio le parti di implementazione e differire i commenti dal codice.
In seguito un esempio del codice PHP della classe immobile:

```
include_once("database.php");
include_once("utente.php");
/**
 * Class: Immobile
 * Description: Classe che implementa un immobile
* @version 1.0
 * @created 16-dic-2015 17:39:15
 */
class Immobile{

	var $database;			//riferimento all'istanza del database utilizzato
	var $id;
	var $approvato;			//valore booleano 0=non approvato 1=approvato
	var $descrizione;
	var $contratto;
	var $immagine;			//URL locale dell'immagine sul server
	var $metratura;			//misura in metri quadrati
	var $prezzo;
	var $proprietario;
	var $tipo;
	var$agente;
	var $comune;

	/**
	 * Function: Immobile
	 * Description: Costruttore della classe Immobile
	 * @param
	*/
	function Immobile(){
		$this->database=new Database();
		$approvato=false;
	}

	//**********************METODI DI ACCESSO**********************//
```

```
	functiongetId(){
		return $this->id;
	}
	functiongetApprovato(){
		return $this->approvato;
	}
	functiongetAgente(){
		return $this->agente;
	}
	functiongetDescrizione(){
		return $this->descrizione;
	}
	functiongetContratto(){
		return $this->contratto;
	}
	functiongetImmagine(){
		return $this->immagine;
	}
	functiongetMetratura(){
		return $this->metratura;
	}
	functiongetPrezzo(){
		return $this->prezzo;
	}
	functiongetProprietario(){
		return $this->proprietario;
	}
	functiongetTipo(){
		return $this->tipo;
	}
	functiongetComune(){
		return $this->comune;
	}
	//**********************METODI MODIFICATORI**********************//
	functionsetApprovato($approvato){
		$this->approvato=$approvato;
	}
	functionsetId($id){
		$this->id=$id;
	}
	functionsetAgente($agente){
		$this->agente=$agente;
	}
	functionsetDescrizione($descrizione){
		$this->descrizione=$descrizione;
	}
	functionsetContratto($contratto){
		$this->contratto=$contratto;
	}
	functionsetImmagine($immagine){
		$this->immagine=$immagine;
	}
	functionsetMetratura($metratura){
		$this->metratura=$metratura;
```

```
	}
	functionsetPrezzo($prezzo){
		$this->prezzo=$prezzo;
	}
	functionsetProprietario($proprietario){
		$this->proprietario=$proprietario;
	}
	functionsetTipo($tipo){
		$this->tipo=$tipo;
	}
	functionsetComune($comune){
		$this->comune=$comune;
	}

	/**
	 * Function: Select
	 * Description: Ricavo i valori di un immobile dal database e li setto nelle variabili di istanza
	 * @param $id id dell'immobile da cui ricavare i dati
	*/
	function select($id){
		$query="SELECT * FROM immobile WHERE id='$id';";
		$this->database->Query($query);
		$result=$this->database->result;
		$row=mysql_fetch_array($result);
		if($row){
			$this->id=$row[0];
			$this->approvato=$row[1];
			$this->agente=$row[2];
			$this->contratto=$row[3];
			$this->descrizione=$row[4];
			$this->proprietario=$row[5];
			$this->immagine=$row[6];
			$this->metratura=$row[7];
			$this->prezzo=$row[8];
			$this->tipo=$row[9];
			$this->comune=$row[10];
		}
	}

	/**
	 * Function: Update
	 * Description: Aggiorna i valori di un immobile già esistente nel database
	 * @param $id id dell'immobile da aggiornare, i valori nuovi saranno ricavati dalle variabili di istanza
	 */
	function update($id){
		$query="UPDATE immobile SET id='$this->id',approvato='$this->approvato',agente='$this-
>agente',tipoContratto='$this->contratto',descrizione='$this->descrizione',proprietario='$this-
>proprietario',immagine='$this->immagine',metratura='$this->metratura',prezzo='$this->prezzo',tipo='$this-
>tipo',comune='$this->comune' WHERE id='$id';";
		$this->database->query($query);
		$risultato=$this->database->result;
		return $risultato;
	}
```

```
        /**
         * Function: Insert
         * Description: Inserisce un nuovo immobile nel database, usando i valori già settati nelle variabili di istanza
         * @param
         */
        functioninsert(){
                $query="INSERT INTO immobile(id, approvato, agente, tipoContratto, descrizione, proprietario, immagine, metratura, prezzo, tipo,comune) VALUES (NULL,'$this->approvato',NULL,'$this->contratto','$this->descrizione','$this->proprietario','$this->immagine' ,'$this->metratura' ,'$this->prezzo' ,'$this->tipo','$this->comune')";
                $this->database->query($query);
                return $this->database->result;
        }

        /**
         * Function: Delete
         * Description: Elimina un immobile dal database
         * @param $id id dell'immobile da eliminare
        */
        function delete($id){
                $query="DELETE FROM immobile WHERE id='$id';";
                $this->database->query($query);
                return $this->database->result;
        }
}
```

Oltre alla descrizione ed ai commenti nel codice si è cercato di rispettare determinate regole sull'utilizzo dei nomi delle variabili,dei metodi e delle classi.
Per quanto riguarda le classi il nome è composto da una serie di sostantivi concatenati, la cui iniziale è scritta in maiuscolo.
Ciascuna categoria di elementi è separato dalle altre da una riga vuota, come ciascun metodo. Ad esempio dopo ogni gruppo di variabili che servono ad uno specifico scopo si ha una riga vuota.
Per quanto riguarda i nomi delle variabili essi devono essere il più chiari possibili sul cosa debba contenere senza essere prolissi e soprattutto mantenendosi rigorosamente al di sotto degli 14 caratteri a variabile, infatti variabili che non rispettano questi criteri tendono a rendere il codice dispersivo in quanto il loro utilizzo è frequente.
Per quanto riguarda i nomi dei metodi è stata applicata la notazione a cammello che prevede di scrivere più parole di seguito senza spazi e con le lettere iniziali maiuscole, inoltre i nomi vengono scelti in base al metodo che verrà generato in modo da classificare laddove possibile i metodi basilari e renderli tutti facilmente riconoscibili.

Più in generale sono state adottate le seguenti regole:

1) Caratteri consentiti:
   Sono a...z, A...Z, 0...9. Inoltre è possibile utilizzare l'underscore "_", sia nei nomi delle costanti (per separare le parole che compongono il nome), sia eventualmente come prefisso di variabili di istanza che potrebbero servire a scopi "interni" della classe.

2) Notazione a cammello:
   E' stata applicata la notazione a cammello descritta precedentemente, invece della notazione ungherese che prevede l'utilizzo di prefissi per descrivere il tipo di dato.

3) Allineamento:
   Indentare correttamente il codice in modo da renderlo più leggibile, strutturando in blocchi omogenei per contesto.

4) Posizione delle variabili:
   Dichiarare le variabili ad inizio blocco, inmodo da raccogliere in un unico punto tutte le dichiarazioni.

Di seguito sono illustrati degli esempi:
- Nomeclasse (dove Nomeclasse è il nome della classe) per il metodo costruttore.
- getDato (dove Dato è il dato a cui accedere) per i metodi di accesso.
- setDato (dove Dato è il dato da settare) per i metodi modificatori.

1.3 Definition, Acronyms and abbreviations
- DB: DataBase.
- MySQL: Acronimo di Structured Query Language, unsoftware open source per manipolazione di database.
- Root: indica il punto iniziale del file system. Il nome deriva dal riferimento alla sua organizzazione ad albero (quindi alla sua radice) comune a molti filesystem.

1.4 References
- Software Engineering di Ian Sommervilledella Addison Weslley.
- Object Oriented Software Engineering di Bernd Bruegge e Allen H. Dutoitdella Prentice Hall.

2. Packages
Overview
I file del codice sono organizzat iin 2 package. Il package principale è la root del nostro server che ospitai file di interfaccia, al suo interno come sottocartella ci troviamo l'altro package, che ospita i file per l'elaborazione e la manipolazione dei dati.
L'interazione col sistema parte dal file index.php, modulo indice visualizzato da qualsiasi utente che si interfaccia per la prima volta con il sistema.

2.2. Descrizione dei packages
Il sistema è suddiviso in tre livelli:

1) Livello di interfaccia:

Ne fanno parte i file: indexCliente.html, indexAgente.html, indexAdmin.html ed infine index.php

2) Livello di controllo e di storage:
   Le classi in questo livello sono tutte le classi che eseguono in qualche modo manipolazioni sulle entità e sui dati quindi ne fanno parte i file: facade*.php, interfaccia*.php, funzioni.php, immobile.php, trattativa.php, utente.php e autenticazione.php.

L'asterisco vuol significare qualsiasi combinazione di caratteri.

3) Livello di storage dei dati

Questo livello comprende il database ed i dati in esso contenuti, le sue capacità sono di verificare l'autenticazione dell'utente e l'esecuzione di query per il recupero di dati da parte di utenti autorizzati.