Dopo aver stilato il documento di Requirements Analysis (RAD) e il documento di System Design(SDD) in cui vi è una descrizione sommaria di ciò che sarà il nostro sistema, definendo i nostri obiettivi ma tralasciando gli aspetti implementativi, andiamo ora a stilare il documento di Object Design che ha come obiettivo quello di produrre un modello che sia in grado di integrare in modo coerente e preciso tutte le funzionalità individuate nelle fasi precedenti.  
In particolar modo, in tale documento si definiscono le interfacce delle classi, le operazioni, i tipi, gli argomenti e la signature dei sottosistemi definiti nel System Design. Inoltre, sono specificati i trade-off e le linee guida.  
Comprensibilità vs Tempo:Il codice del sistema deve essere comprensibile il più possibile, in modo da facilitare la fase di testing ed eventuali future modifiche da apportare. Per rispettare queste linee guida il codice sarà accompagnato da commenti volti a semplificarne la comprensione.

Ovviamente questo comporterà un aumento del tempo di sviluppo del nostro progetto.  
Prestazioni vs Costi:Dato che il nostro progetto è sprovvisto di budget, per poter mantenere prestazioni elevate, in determinate funzionalità verranno utilizzati dei template open source esterni.

Interfaccia vs Usabilità:  
L’interfaccia grafica è stata realizzata in maniera molto semplice, chiara e concisa, vengono utilizzati i form e pulsanti con lo scopo di rendere semplice l’utilizzo del sistema da parte dell’utente finale.  
Sicurezza vs Efficienza:La sicurezza, come descritto nei requisiti non funzionali del Requirements Analysis, rappresenta uno degli aspetti importanti del sistema. Tuttavia, dati i tempi di sviluppo molto limitati, ci limiteremo ad implementare sistemi di sicurezza basati su username e password degli utenti.  
1.2 Linee Guida per la Documentazione delleInterfacceGli sviluppatori dovranno seguire determinate linee guida per la stesura del codice:  
Naming Convention:È buona norma utilizzare nomi:  
● Descrittivi  
● Pronunciabili  
● Di uso comune  
● Di lunghezza medio-corta  
● Non abbreviati  
● Utilizzando solo caratteri consentiti (a-z, A-Z, 0-9)  
6  
Variabili:● I nomi delle variabili devono iniziare con la lettera minuscola, e le parole successive con la lettera maiuscola. La dichiarazione delle variabili deve essere effettuata ad inizio blocco; in ogni riga vi deve essere una sola dichiarazione di variabile e va effettuato l’allineamento per migliorare la leggibilità.  
Esempio: idProdotto, nomeProdotto  
● In determinati casi, è possibile utilizzare il carattere underscore “\_”, ad esempio quando si fa uso di variabili costanti oppure quando si fa uso di proprietà statiche.  
Esempio: AGGIUNGI\_PRODOTTO  
Metodi:● I nomi dei metodi devono iniziare con la lettera minuscola, e le parole successive con la lettera maiuscola. Di solito il nome del metodo è costituito da un verbo che identifica un’azione, seguito dal nome di un oggetto.  
I nomi dei metodi per l’accesso e la modifica delle variabili dovranno essere del tipo getNomeVariabile() e setNomeVariabile(). Se viene dichiarata una variabile all’interno di un metodo quest’ultima deve essere dichiarata appena prima del suo utilizzo e deve servire per un solo scopo, per facilitare la leggibilità.  
Esempio: getId(), setId()  
● Ai metodi va aggiunta una descrizione, la quale deve essere posizionata prima della dichiarazione del metodo, e deve descriverne lo scopo.

La descrizione del metodo deve includere anche informazioni riguardanti gli argomenti, il valore di ritorno, le  
eccezioni. I metodi devono essere raggruppati in base alla loro funzionalità.  
Classi e pagine:● I nomi delle classi e delle pagine devono iniziare con la lettera maiuscola, e anche le parole successive all’interno del nome devono iniziare con la lettera maiuscola.  
I nomi delle classi e delle pagine devono essere evocativi, in modo da fornire informazioni sullo scopo di quest’ultime.  
Esempio: GestioneFornitori.java  
Ogni file sorgente .java contiene una singola classe e dev’essere strutturato in un  
determinato modo:  
● Una breve introduzione alla classe  
L’introduzione indica: l’autore, la versione e la data.  
/\*\*  
\* sommario dello scopo della classe.  
\*  
\* @author [nome dell’autore]  
\* @version [numero di versione della classe]  
\*/  
● L’istruzione include che permette di importare all’interno della classe gli altri oggetti  
che la classe utilizza.  
● La dichiarazione di una classe è caratterizzata da:  
● Dichiarazione della classe pubblica  
● Dichiarazioni di costanti  
● Dichiarazioni di variabili di classe  
● Dichiarazioni di variabili d’istanza  
● Costruttore  
● Commento e dichiarazione metodi e variabili

ESEMPIO:  
class Prodotto {  
/\*\*  
\* @var int id del prodotto  
\*/  
private id;  
/\*\*  
\* @var string nome il nome del prodotto  
\*/  
private nome;  
/\*\*  
\* Costruttore di Prodotto  
\* @param string id L’identificativo del prodotto  
\* @param string nome Il nome del Prodotto  
\*/  
public function \_construct(id\_2, nome\_2) {  
this.id = id\_2;  
this.nome = nome\_2;  
}  
/\*\*  
\* restituisce l’id del prodotto  
\* @return string L’id del Prodotto  
\*/  
public function getId() {  
return this.id;  
}  
/\*\*  
\* restituisce il nome del prodotto  
\* @return string Il nome del Prodotto  
\*/  
public function getNome() {  
return this.nome;  
}  
/\*\*  
\* Setta l’id del Prodotto  
\* @param string id L’id del prodotto  
\*/  
public function setId(id\_2) {  
this.id = id\_2;  
}  
/\*\*  
\* Setta il nome del Prodotto  
\* @param string nome Il nome del Prodotto  
\*/  
public function setNome(nome\_2) {  
this.nome = nome\_2;  
}

}

**2.1 Design pattern globali**

**Command Design Pattern**: Letsmeet fa uso del Command Pattern che è uno dei pattern fondamentali, che permette di isolare la porzione di codice che effettua un'azion (eventualmente molto complessa) dal codice che ne richiede l'esecuzione; l'azione è incapsulata nell'oggetto Command. L'obiettivo è rendere variabile l'azione del client senza però conoscere i dettagli dell'operazione stessa.

**Singleton Design Pattern**: LetsMeet fa uso del Singleton Pattern nello Storage Layer per impedire la creazione di più di una istanza di oggetto di tipo Manager.

**3.0 Package Component**

La gestione del nostro sistema è suddivisa in tre livelli (three-tier):

● Interface layer

● Application Logic layer

● Storage layer

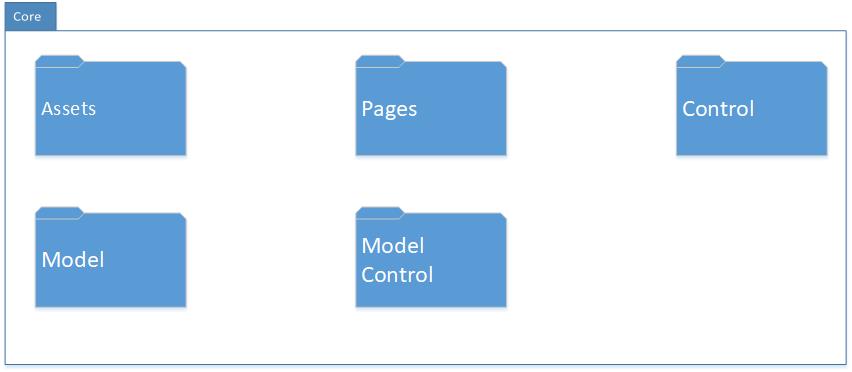
Il package “LetsMeet” contiene sottopackage che a loro volta inglobano classi atte alla gestione delle richieste utente. Le classi contenute nel package svolgono il ruolo di gestore logico del sistema.

|  |  |
| --- | --- |
| Interface layer | Rappresenta l’interfaccia del sistema, ed offre la possibilità all’utente di interagire con quest’ultimo, offrendo sia la possibilità di inviare, in input, che di visualizzare, in output, dati. |

|  |  |
| --- | --- |
| Application Logic layer | Ha il compito di elaborare i dati da inviare al client, e spesso grazie a delle richieste fatte al database, tramite lo Storage Layer, accede ai dati persistenti.  Si occupa di varie gestioni quali:  1. Gestione Account  2. Gestione Eventi  3. Gestione Segnalazione |

|  |  |
| --- | --- |
| Storage layer | Ha il compito di memorizzare i dati persistenti del sistema, utilizzando un DBMS, inoltre riceve le varie richieste dall’ Application Logic layer inoltrandole al DBMS e restituendo i dati richiesti. |

* **Assets:** contiene tutti i file multimediali dell’applicazione.
* **Pages:** contiene le pagine dell’applicazione e ognuna di esse ha una cartella dove vi è definita la pagina (.html o .jsp), lo stile (.css) e l’eventuale script (.js).
* **Control:** contiene tutte le classi che permettono di gestire la logica di controllo dell’applicazione.
* **Model:** contiene tutte le classi che permettono la manipolazione dei dati persistenti dell’applicazione.
* **Model Conection:** contiene tutte le classi che permettono la connessione al DBMS.



**3.0.1 Package Model**

****

|  |  |
| --- | --- |
| Classe | Descrizione |
| Utente.java | Descrive l’utente registrazione |
| Evento.java |  |
| Segnalazione.java |  |