

ODD

Object Design Document

MedQueue

Afeltra Angelo

Amato Adriano

Fucile Andrea

Rapa Giovanni

Sommario

1. **Introduzione**
   1. Object design trade-offs
      1. Componenti off-the-shelf
      2. Design pattern
   2. Linee guida per la documentazione dell’interfaccia
   3. Definizioni, acronimi e abbreviazioni
   4. Riferimenti
2. **Packages**
   1. Divisione in pacchetti
   2. Organizzazione del codice in file
3. **Interfacce delle classi**
4. **Class diagram**
5. **Glossario**
6. **INTRODUZIONE**

Dopo la realizzazione dei documenti RAD e SDD abbiamo descritto in linea massima , quello che sarà il nostro sistema e quindi i nostri obiettivi, tralasciando gli aspetti d’implementazione.

Il seguente documento ha lo scopo di produrre un modello capace di integrare in modo coerente e preciso tutte le diverse funzionalità individuate nelle fasi precedenti.

In particolare, si vanno a descrivere i trade-offs che dovranno essere rispettati dagli sviluppatori, i design pattern stabiliti, le linee guida sulla documentazione delle interfaccie, la divisione in pacchetti e le interfaccie delle classi da sviluppare.

* 1. **Object design trade-offs**

**Leggibilità vs Tempo**

Il codice deve essere al quanto più comprensibile per poter facilitare la fase di testing ed eventuali future modifiche.

A tale scopo andranno inseriti commenti nel codice che ne specificano la comprenzione.

Questa caratteristica aumenta il tempo di sviluppo ma allo stesso tempo lo rende piu comprensibile

**Modificabilita vs Costi**

La necessità di sviluppare un’applicazione web apre alla possibiltà di utilizzare una vastissima collezione di framework e librerie utili per la realizzazione del prodotto.

Data la facilita di adattamento del team di sviluppo all’utlizzo di nuove componenti abbiamo adottato l’utilizzo di componenti off-the-shelf per il core dell’applicazione.

Questa caratteristica aumenta la modificabilità del prodotto e allo stesso tempo utilzzando framework free non inciderà sui cosi.

* + 1. **Componenti off-the-shelf**

Per motivazioni legate al budget ridotto, si ricorrerà all’utilizzo di un framework per la realizzazione dell’interfaccia utente: Ionic.

Ionic Framework si pone come supporto per la creazione d’interfaccie grafiche per applicazioni ibride, i loro autori hanno raccolto le miglori “best practice” per lo sviluppo di interfacce mobile con tecnologie Web e le hanno codificate in questo framework, evitando quindi che ogni sviluppatore riparta da zero nello sviluppo dell’interfaccia di una nuova applicazione.

Questa soluzione si presta alla realizzazione di un’interfaccia completa e minimale, ideale per il servizio che forniremo.

Il back-end farà invece forte affidamento sul framework Spring, nota soluzione nell’ambito delle applicazioni distribuite Java. Composto da un core ben ottimizzato e prodotto con l’obiettivo di ridurre quello che il codice “boilerplate”, consente agli sviluppatori di concentrarsi maggiormente sulla logica di business dell’applicazione, piuttosto che sulla comunicazione tra le varie componenti. Il modulo d’interesse per questo progetto è Spring MVC, framework per la realizzazione di applicazioni web basate sul modello MVC che permettere di scrivere codice riusabile.

* + 1. **Design patterns**

Per velocizzare lo sviluppo della webapp, e anticipare i possibili cambiamenti del sistema si utilizza:

**1.1.2.1 MVC**

Model-view-controller, è un pattern architetturale molto diffuso nello sviluppo di sistemi software, in particolare nell'ambito della programmazione orientata agli oggetti e in applicazioni web.

Utilizzando l’mvc avremo un applicazione modulare divisa in tre tipologie di componenti:

* Model: che implementa le funzionalità di business
* View che implementa la logica di presentazione
* Controller: che implementa la logica di controllo

**1.1.2.2 Design Pattern DAO**

Il DAO (Data Access Object) è un pattern architetturale per la gestione della persistenza: si tratta fondamentalmente di una classe con relativi metodi che rappresenta un'entità tabellare di un RDBMS, usata principalmente in applicazioni web per stratificare e isolare l'accesso ad una tabella tramite query.

I metodi del DAO con le rispettive query dentro verranno così richiamati dalle classi della business logic.

Il vantaggio relativo all'uso del DAO è dunque il mantenimento di una rigida separazione tra le componenti di un'applicazione, le quali potrebbero essere il "Modello" e il "Controllo" in un'applicazione basata sul paradigma MVC.

* 1. **Linee guida per la documentazione dell’interfaccia**

E richiesto agli sviluppatori di seguire le seguenti linee guida al fine di essere consistenti nell’intero progetto e facilitare la comprensione delle funzionalità di ogni componente.

Nomenclatura delle componenti:

* Nomi delle classi
* Ogni classe deve avere nome in CamelCase
* Ogni classe deve avere nome singolare
* Ogni classe che modella un’entità deve avere per nome un sostantivo che possa associarlo alla corrispondente entità di dominio
* Ogni classe che realizza un servizio offerto via web deve avere nome composto da nome del servizio seguito dal suffisso “Control”
* Nomi dei metodi
* Ogni metodo deve avere nome in lowerCamelCase
* Ogni metodo in cui si può verificare un errore deve segnalare l’errore sollevando un’eccezione
* Nomi delle eccezioni
* Ogni eccezione deve avere nome esplicativo del problema segnalato
* Nomi degli altri sorgenti
* Ogni documento JSP deve avere nome che possa ricondurre al contenuto da essa mostrato

Organizzazione delle componenti:

* Tutte le classi che realizzano un sottoinsieme devono essere racchiuse nello stesso pacchetto Java
* Tutte le componenti che realizzando l’interfaccia grafica devono essere collocate nella directory “Inserire directory”
* Tutte le risorse statiche (fogli di stile, script e immagini) devono essere collocate nella directory “resources”

Organizzazione del codice:

* Il codice Java dev’essere indentato in maniera appropriata (ese: il checkstyle di Google)
* Il codice HTLM dev’essere indentato in maniera appropriata (ese: il checkstyle di Google)
  1. **Definizione, acronimi e abbreviazioni**

CheckStyle: strumento di analisi statica del codice utilizzato nello sviluppo software per verificare se il codice sorgente Java è conforme alle regole di codifica specificate.

* 1. **Riferimenti**
* Design goals: sezione 1.2 dell’SDD
* Scelta dell’ambiente d’esecuzione: sezione 3.2 e 3.3 dell’SDD
* Spring: [Spring](https://spring.io/)
* Ionic: [Ionic](https://ionicframework.com/)

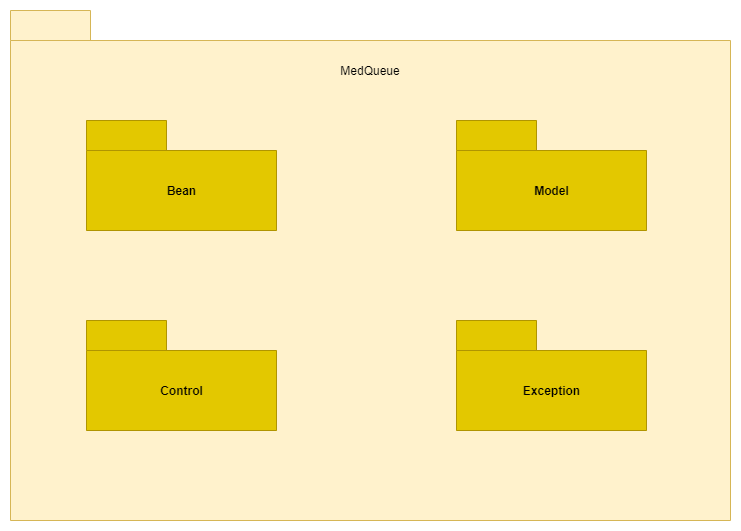
1. **Packages**

In questa sezione presentiamo in modo più approfondito quella che è la divisione in sottosistemi e l’organizzazione del codice in file

* 1. **Divisione in pacchetti**

Il sistema rispecchia l’architettura definita nel documento SDD, qundi avremo un sistema suddiviso in tre livelli (three-tier):

|  |  |
| --- | --- |
| Presetazione Layer | Rappresenta l’interfaccia del sistema, ed offre la possibiltà all’utente di interagire con quest’ultimo, offrendo sia la possibiltà di inviare, in input, che di visualizzare in output dati |
| Business Layer | Ha il compito di elaborare i dati da inviare al client, e spesso graazie a delle richieste fatte al database, tramite lo Persistene Layer, accede ai dati persistenti.  Si occupa di varie gestioni quali:   * Accesso * Prenotazione * Visualizzazione Coda * Gestione |
| Persistence Layer | Ha il compito di memorizzare i dati sensibili del sistema utilizzando un DBMS, inoltre riceve le varie richieste dal Business Layer inoltrandole al DBMS e restituendo i dati richiesti |



**2.1 Package Bean**

****

|  |  |
| --- | --- |
| Classe: | Descrizione |
| AmbulatorioBean | Questa classe rappresenta un ambulatorio della struttura ospedaliera |
| ImpiegatoBean | Questa classe rappresenta un impiegato della struttura ospedaliera |
| PrenotazioneBean | Questa classe rappresenta la prenotazione dell’utente |
| StrutturaBean | Questa classe rappresenta la struttura ospedaliera |
| OperazioneBean | Questa classe rappresenta le operazioni per cui si puo prendere una prenotazione |
| UtenteBean | Questa classe rappresenta l’utente registrato |

* 1. **Package Control**

****

**2.3 Package Model**