Università degli Studi di Salerno
Corso di Ingegneria del Software

Digiworld
Object Design Document

Data: GG/MM/AAAA

Coordinatore del progetto:

Nome	Matricola
Jacopo Sabino Sole	0512106112

Partecipanti:

Nome	Matricola
Salvatore Senarcia	0512106100
Vincenzo Fortino	0512101240
Sabato Pescatore	0512105800
Jacopo Sabino Sole	0512106112

Revision History

Data	Versione	Descrizione	Autore

Sommario

1. Introduzione

1.1 Object design trade-offs

- Considerata l'ingente esistenza di framework e gli esigui e stringenti tempi per la consegna del sistema, partendo da un'attenta disamina di strumenti già esistenti, legata al riciclo e riutilizzo di codice, abbiamo deciso di fare affidamento il più possibile a componenti off-the-shelf per il core ed a soluzioni offerte da terzi, scegliendo con attenzione, però, quali tra queste possano fare al caso nostro.
- Tutto questo è stato possibile grazie all'estrema flessibilità e dinamicità del gruppo di lavoro.

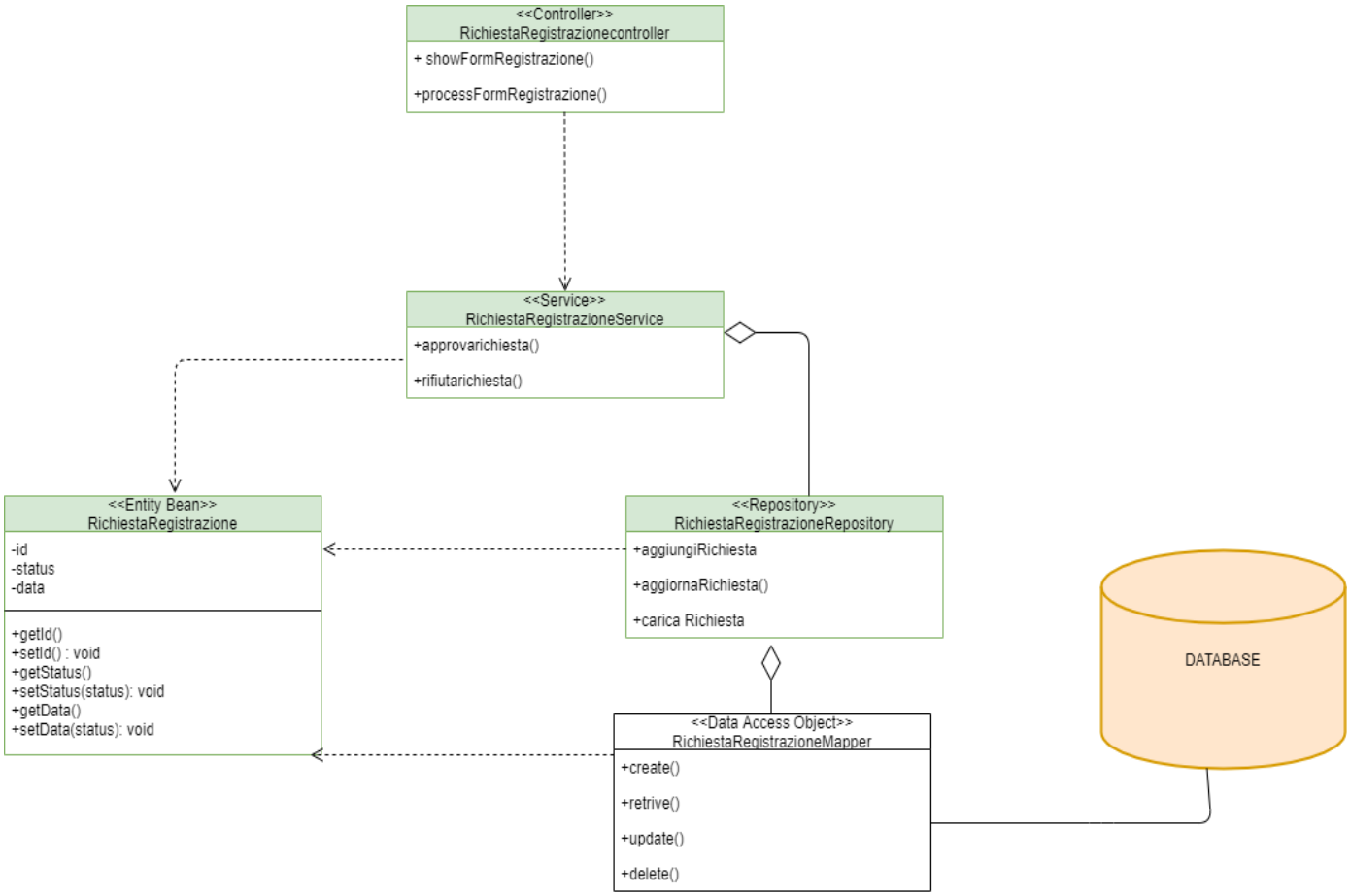
1.1.1 Componenti off-the-shelf

- Per la realizzazione dell'interfaccia utente la nostra scelta è ricaduta sul framework **bootstrap**, compatibile con le ultime versioni di tutti i principali browser, che grazie ai suoi modelli di progettazione basati su HTML e CSS. Inoltre possiamo facilmente realizzare il responsive web design grazie al quale il layout delle pagine web si regola dinamicamente, tenendo conto delle caratteristiche del dispositivo utilizzato, sia esso desktop, tablet o telefono cellulare.

1.1.2 Design Patterns

- Abbiamo utilizzato alcuni design patterns che risolvono problemi comuni presenti nel nostro

progetto.



1.1.2.1 Data Mapper

Il data mapper è un modello sostenuto da Martin Fowler, il cui scopo è separare gli oggetti che rappresentano le informazioni persistenti dalla logica relativa alla mappatura di tali informazioni sul database sottostante. Ciò consente agli oggetti che risiedono nella memoria di essere completamente separati dal livello sottostante (che non ha nulla a che fare con database relazionali, file o altri livelli), facilitando così la manutenibilità del sistema.

1.1.2.2 Repository pattern

Questo modello è stato proposto da Edward Hieatt e Rob Mee e promuove la definizione di uno strato intermedio tra oggetti che implementano la logica di business e oggetti che accedono a dati persistenti. In questo modo, l'applicazione ha un'interfaccia attraverso la quale gli oggetti persistenti possono essere considerati come oggetti allocati in una normale raccolta, e sull'interfaccia possono essere eseguite operazioni banali come l'aggiunta, l'eliminazione, l'aggiornamento e la ricerca: gli oggetti che interagiscono con questa collezione non sono a conoscenza di ciò che accade dietro le quinte, in quanto è proprio la repository ad occuparsi della propagazione dell'informazione alla sorgente di dati persistente. È il repository stesso, che incapsula tutte le possibili operazioni di ricerca in una raccolta di oggetti: di conseguenza, anche le operazioni sugli oggetti persistenti che risiedono nei database relazionali sono più orientate agli oggetti.

1.1.2.3 Service Layer

A volte, le applicazioni che sviluppiamo forniscono le loro funzioni attraverso interfacce diverse: ognuna

fornisce un insieme di operazioni disponibili in modo diverso, ma tutte queste interfacce hanno una logica applicativa. Pertanto, Randy Stafford ha avanzato un suggerimento: separare completamente la logica di presentazione dalla logica di business e inserire la logica presentazione in un livello per esporre tutti i servizi che l'applicazione deve fornire. Il risultato, quindi, è che la logica dell'applicazione è completamente indipendente dall'applicazione del metodo di presentazione: ne consegue che l'applicazione è più semplice da mantenere ed estendere attraverso nuovi servizi e interfacce.

1.2 Linee guida per la documentazione d'interfaccia

È 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 associarla alla corrispondente entità di dominio
- Ogni classe che realizza la logica di business per una determinata entità deve avere nome composto da quello del pacchetto per cui espone servizi seguito da "Service"
- Ogni classe che modella una collezione in memoria per una determinata classe di entità deve avere nome composto da quello dell'entità su cui opera seguito da "Repository"
- Ogni classe che realizza un form deve avere nome composto dal sostantivo che descrive il form seguito dal suffisso "Form"
- Ogni classe che realizza un servizio offerto via web deve avere nome composto dal nome del pacchetto per cui espone servizi seguito dal suffisso "Controller"

Nomi dei metodi

- Ogni metodo deve avere nome in lowerCamelCase
- Ogni metodo deve avere nome che inizia con un verbo in seconda persona singolare
- Ogni metodo deve segnalare un 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 sottosistema devono essere racchiuse nello stesso pacchetto Java
- Tutte le componenti che realizzano l'interfaccia grafica devono essere collocate nella directory "/WEB-INF/views/<pertinenza>/" a seconda della tipologia di componente di interfaccia realizzata. <pertinenza> può essere uno tra "pages", "forms", "common", "lists" e "menu".
- 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 (tramite 2 spazi, non tabulazioni) e

l'apertura di un blocco di codice deve avvenire nella stessa riga in cui è specificato il nome della classe o la firma del metodo che quel blocco definisce

- Il codice HTML dev'essere indentato in maniera appropriata (tramite 2 spazi, non tabulazioni), gli elementi vuoti non devono essere chiusi (
 invece di
) e gli attributi devono essere indicati in minuscolo.

1.2 Riferimenti

N/A

2. Packages

Qui spieghiamo dettagliatamente la divisione in sottosistemi e l'organizzazione del codice n

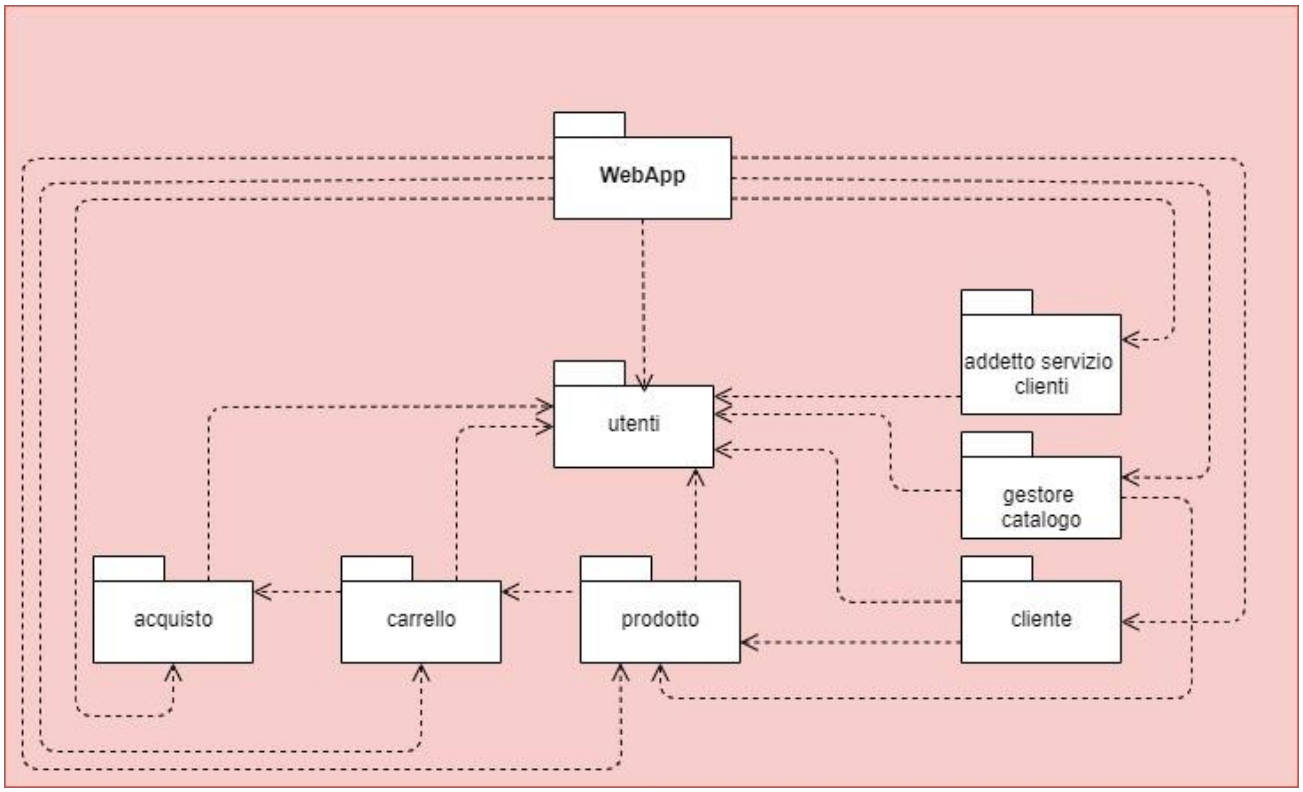
2.1 Divisione in pacchetti

Il sistema è diviso in 3 layers con determinati sottosistemi e per ognuno di essi creeremo un pacchetto. Questo ci serve a mettere insieme le classi legate tra loro e facendo in modo che il tutto sia diviso verticalmente e non ci siano eccessive interdipendenze tra i pacchetti.

L'unico sottosistema a non essere suddiviso in pacchetti è quello di accesso ai dati persistenti. Questi ultimi verranno gestiti dalla piattaforma MySQL Workbench. Il team di programmazione provvederà alla connessione tra il sistema ed il database esterno ma non dovrà poi intervenire su questa sezione.

Riportiamo in seguito lo schema con l'intera decomposizione in pacchetti e la descrizione

di questi ultimi.



WebApp	
Descrizione	Definisce le classi necessarie all'avvio dell'applicazione e le configurazioni della suddetta ed include tutti gli altri pacchetti
Dipendenze	N/A

Utenti	
Descrizione	Definisce le proprietà dell'utente generico della piattaforma ed espone i servizi di autenticazione, logout e ottenimento dell'utente autenticato
Dipendenze	

Addetto Servizio Clienti	
Descrizione	Definisce le proprietà dell'ADS ed espone i servizi per la manipolazione dei commenti e delle richieste di aiuto fatte tramite e-mail.
Dipendenze	Utenti

Gestore Catalogo

Descrizione	Definisce le proprietà del gestore del catalogo ed espone i servizi per la manipolazione dei prodotti nel catalogo e la loro aggiunta, modifica e rimozione.
Dipendenze	Utenti

Prodotto	
Descrizione	Definisce le proprietà dei prodotti ed espone i servizi per la manipolazione degli stessi.
Dipendenze	

Carrello	
Descrizione	Definisce le proprietà del carello ed espone i servizi per la manipolazione degli prodotti al suo interno.
Dipendenze	Prodotto

Acquisto	
Descrizione	Definisce le proprietà dell'acquisto ed espone i servizi per la manipolazione dei prodotti acquistati.
Dipendenze	Prodotto, Carello

2.2 Organizzazione del codice in file

Come imposto da Java, ogni classe del sistema sarà collocata nel relativo file. Ognuno di essi sarà quindi collocato nella cartella dedicata (individuata in base al nome del pacchetto).

I nomi dei pacchetti avranno tutti come prefisso it.unisa.di.digiworld (e saranno mappati nel rispettivo percorso src/main/java/it/unisa/di/…) ad eccezione del pacchetto realizzante l'interfaccia utente, che sarà invece collocato nella directory src/main/webapp/WEB-INF/views/.

Si farà inoltre ricorso alla directory src/main/resources per la definizione di configurazioni e messaggi.

3. Interfacce delle classi

Per accedere alla documentazione si consiglia di accedere tramite il file index.html

4. Class diagram

5. Glossario

Componenti off-the-shelf: prodotti software sviluppati da terzi riutilizzabili**.**

CSS: Linguaggio per la definizione degli stili delle pagine web.

Framework: Software di supporto allo sviluppo**.**

HTML: Linguaggio per la strutturazione delle pagine web.

JavaScript: Linguaggio di scripting nato per dare dinamicità alle pagine HTML.

Material Design: insieme di linee guida stilate da Google per il design di interfacce grafiche.

Codice boilerplate: Sezione di codice ripetuta in più punti senza alterazioni.

JavaServer Pages: Tecnologia che facilita lo sviluppo di pagine web dinamiche fornendo un'interfaccia Java tagoriented con il back end.

Javadoc: Documentazione generata automaticamente a partire dai commenti scritti nei sorgenti**.** **Java**