TirocinioSmart

ODD
Object Design Document

TirocinioSmart

Aniello Carotenuto
Giuseppe Dello Stretto
Rosanna Coccaro

Data	Versione	Cambiamenti	Autori
02/12/2017	0.1	Stesura introduzione	AC
13/12/2017	0.3	Aggiornamento introduzione e definizione packages	AC
13/12/2017	0.4	Aggiunta class interfaces	RC
13/12/2017	0.5	Aggiunta class diagram	GDS
15/12/2017	1.0	Correzioni varie	[tutti]
27/12/2017	1.1	Aggiornamento class diagram	AC
03/02/2018	1.2	Revisione finale	AC

Sommario

1. Introduzione

1.1 Object design trade-offs

Buy vs Build: la necessità di sviluppare un'applicazione web apre alla possibilità di attingere da una vastissima collezione di framework e librerie utili alla realizzazione del prodotto. La volontà di non voler reinventare nuovamente la ruota ed i tempi stringenti per la consegna fanno quindi valutare l'adozione di componenti off-the-shelf per il core dell'applicazione. La linea guida, in questo caso, è quindi quella di riutilizzare il più possibile le soluzioni offerte da terzi, scegliendo con attenzione, però, quali tra queste possano fare al caso nostro.

È inoltre rilevante sottolineare che le scelte sono state dettate anche dalla facilità di adattamento del team di sviluppo alle nuove componenti: si è infatti cercato il compromesso tra la necessità di familiarizzare con una tecnologia sconosciuta e la reale utilità della suddetta.

1.1.1 Componenti off-the-shelf

Per motivazioni legate al budget ridotto, si ricorrerà all'utilizzo di un framework per la realizzazione dell'interfaccia utente: **materializecss**. Scritto in HTML, CSS e JavaScript, questo framework ha l'obbiettivo di portare anche su servizi web quello che va sotto il nome di "Material Design", sviluppato da Google con l'obbiettivo di fornire linee guida precise per la relizzazione di un'interfaccia grafica usabile. Questa soluzione ben 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'obbiettivo 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. I moduli d'interesse per questo progetto saranno Spring MVC (che fornisce tutto il meccanismo di controllo del flusso) e Spring Data JPA (ORM).

La comunicazione tra front-end e back-end ricorrerà all'utilizzo delle **JavaServer Pages** e della libreria **JSTL** per la composizione dinamica delle pagine che verranno poi presentate all'utente.

Tutte le componenti selezionate sono gratuite ed open source, scelta in linea con i requisiti di costo.

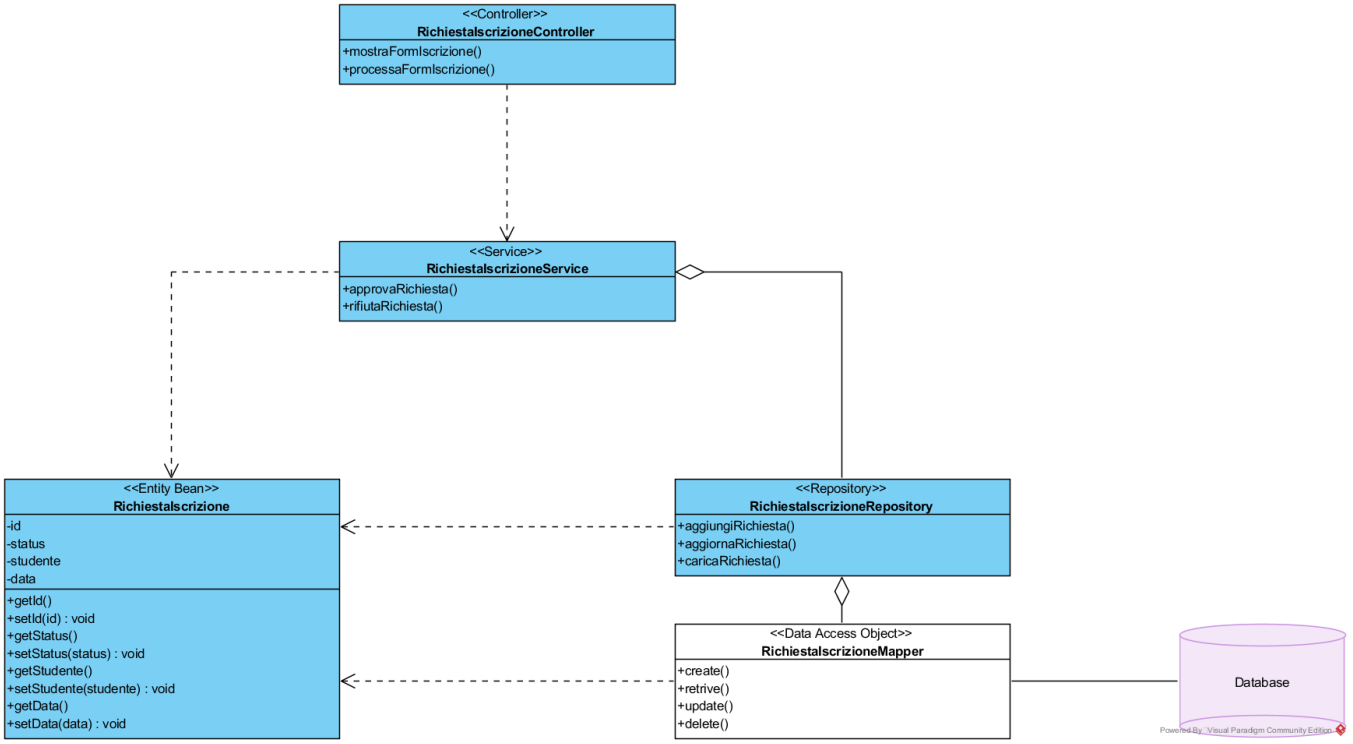
1.1.2 Design patterns

Per velocizzare lo sviluppo del sistema senza dover "reinventare la ruota" ogni volta, abbiamo utilizzato alcuni design patterns che risolvono problemi comuni presenti anche nel nostro progetto. Questi pattern non sono tra quelli proposti dalla GoF ma riguardano specificamente le applicazioni enterprise.

I pattern adottati si integrano correttamente e, pertanto, presentiamo il diagramma che mostra come si relazionano le varie componenti introdotte mentre rimandiamo ai link presenti nella sezione 1.4 per schemi dettagliati che riguardano i singoli pattern.

Precisiamo che le componenti mostrate nel seguente diagramma non rappresentano componenti reali del sistema, ma dei semplici oggetti che hanno lo scopo di semplificare la rappresentazione della struttura per facilitarne la comprensione.

Segnaliamo che l'oggetto mapper ha un colore differente dagli altri perché, come riportato inoltre nella sezione 2, la sua implementazione non sarà a carico del programmatore.



1.1.2.1 Data mapper

Il Data Mapper è un pattern promosso da Martin Fowler con l'obbiettivo di separare gli oggetti che rappresentano l'informazione persistente dalla logica relativa alla mappatura di questa informazione sul database sottostante. Ciò permette agli oggetti che risiedono in memoria di essere completamente disaccoppiati dal livello sottostante (indipendentemente da quale esso sia, database relazionale, file o altro), favorendo la manutenibilità del sistema.

1.1.2.2 Repository pattern

Proposto da Edward Hieatt e Rob Mee, questo pattern promuove la definizione di un livello intermedio tra gli oggetti che realizzano la logica di business e quelli che accedono ai dati persistenti. In questo modo, si fornisce all'applicazione un'interfaccia tramite cui gli oggetti persistenti possono essere visti come allocati in una normale collezione su cui è possibile effettuare operazioni banali di aggiunta, rimozione, aggiornamento e 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 la repository stessa ad incapsulare tutte le possibili operazioni di ricerca all'interno della collezione di oggetti: il risultato è una visione maggiormente orientata agli oggetti anche per le operazioni su oggetti persistenti che risiedono su database relazionali.

1.1.2.3 Service Layer

Talvolta le applicazioni che sviluppiamo offrono le proprie funzionalità tramite interfacce differenti: ognuna di esse presenta l'insieme di operazioni disponibili in modo differente ma la logica applicativa è comune a tutte queste interfacce. Da ciò la proposta di Randy Stafford: separare completamente la logica di presentazione da quella di business, collocando quest'ultima in un livello che espone tutti i servizi che l'applicazione ha da offrire.

Il risultato è quindi quello di un'applicazione la cui logica applicativa è completamente indipendente dalla modalità di presentazione: segue che l'applicazione è maggiormente manutenibile ed estendibile con nuovi servizi ed interfacce.

Tramite il livello di servizio abbiamo quindi la definizione di oggetti che espongono tutti i servizi offerti dal relativo sottosistema, in grado di realizzare transazioni su multiple entità di dominio ed accentrare i controlli di sicurezza.

1.2 Linee guida per la documentazione dell'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

1.3 Definizioni, acronimi e abbreviazioni

N/A

1.4 Riferimenti

- Design goals: sezione 1.2 dell'SDD

- Scelta dell'ambiente d'esecuzione: sezioni 3.2 e 3.3 dell'SDD
- materializecss: documentazione
- Spring: documentazione
- Google coding style: Java, HTML/CSS
- Design patterns: Data Mapper, Repository, Service Layer

2. Packages

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

2.1 Divisione in pacchetti

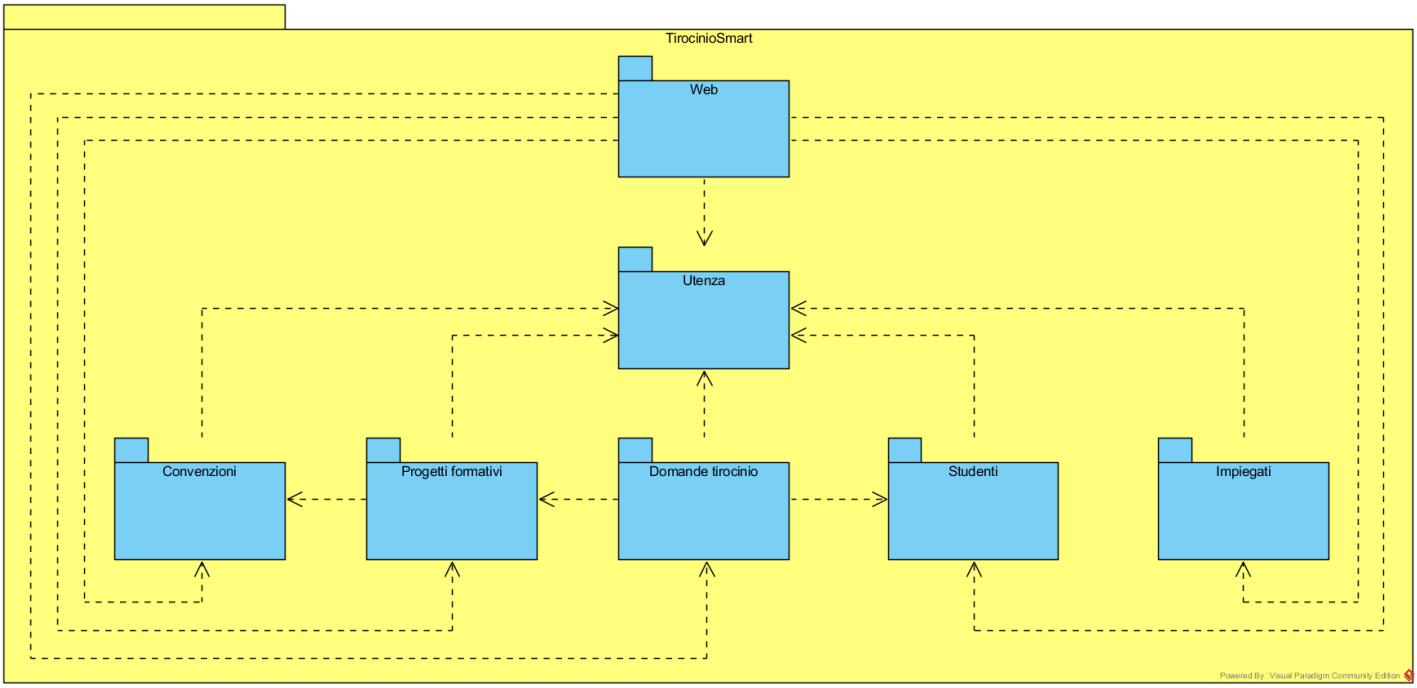
La divisione del sistema in pacchetti condivide molti aspetti con quella proposta nel documento di System Design: il sistema è diviso in 3 layer, ognuno dei quali contiene un discreto numero di sottosistemi e per ognuno di essi sarà realizzato un pacchetto, tranne che per il sottosistema di accesso ai dati persistenti.

Questa divisione nasce dalla volontà di riunire nello stesso pacchetto le classi che sono legate da una forte dipendenza da un punto di vista semantico: la divisione è stata infatti verticale, eccezion fatta per l'interfaccia utente, in modo da ridurre le interdipendenze tra pacchetti, che esistono tuttavia a causa della ripartizione degli oggetti di dominio in pacchetti diversi (ci sarebbe stato un problema simile dividendo in modo orizzontale, dato che ogni controller sarebbe dipeso comunque dagli oggetti di dominio).

Quanto al sottosistema di accesso ai dati, questo non sarà implementato dagli sviluppatori e non comparirà nei sorgenti del sistema per via della soluzione utilizzata per l'implementazione: i vari pacchetti di gestione conterranno infatti delle interfacce (chiamate "repository") che specificano i metodi che dovranno offrire le classi di accesso al database. La concretizzazione di tali interfacce, tuttavia, è a carico del framwork adottato e non richiede l'intervento del programmatore, che dovrà limitarsi a specificare i parametri di connessione nel file application.properties.

Le classi per la configurazione, assieme a quelle di servizio, saranno collocate nel pacchetto webapp.

Riportiamo di seguito lo schema di decomposizione in sottosistemi e la descrizione, comprensiva di dipendenze, per ognuno dei pacchetti.



Per una migliore comprensione, si evidenzia che il pacchetto che realizza l'interfaccia web dell'applicazione dipende da tutti e 6 i pacchetti di logica applicativa e che tra questi ultimi il pacchetto utenza ricopre un ruolo centrale.

Nome pacchetto	Utenza
Descrizione	Definisce le proprietà dell'utente generico della piattaforma ed espone i servizi di autenticazione, logout e ottenimento dell'utente autenticato
Dipendenze	N/A

Nome pacchetto	Convenzioni
Descrizione	Definisce le proprietà di aziende e richieste di convenzionamento ed espone i servizi per la manipolazione delle richieste e la visualizzazione delle informazioni sulle aziende
Dipendenze	Utenza

Nome pacchetto	Studenti
Descrizione	Definisce le proprietà di studenti e richieste d'iscrizione ed espone i servizi per la manipolazione delle richieste e la visualizzazione delle informazioni sugli studenti
Dipendenze	Utenza

Nome pacchetto	Impiegati
Descrizione	Definisce le proprietà dei delegati aziendali ed espone i servizi per la visualizzazione delle informazioni su di essi
Dipendenze	Utenza

Nome pacchetto	Progetti formativi
Descrizione	Definisce le proprietà dei progetti formativi ed espone i servizi per l'aggiunta e l'archiviazione degli stessi
Dipendenze	Convenzioni

Nome pacchetto	Domande tirocinio
Descrizione	Definisce le proprietà delle domande di tirocinio ed espone i servizi per l'accettazione, il rifiuto, l'approvazione ed il respingimento delle suddette
Dipendenze	Progetti formativi, Studenti

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

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.tirociniosmart (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

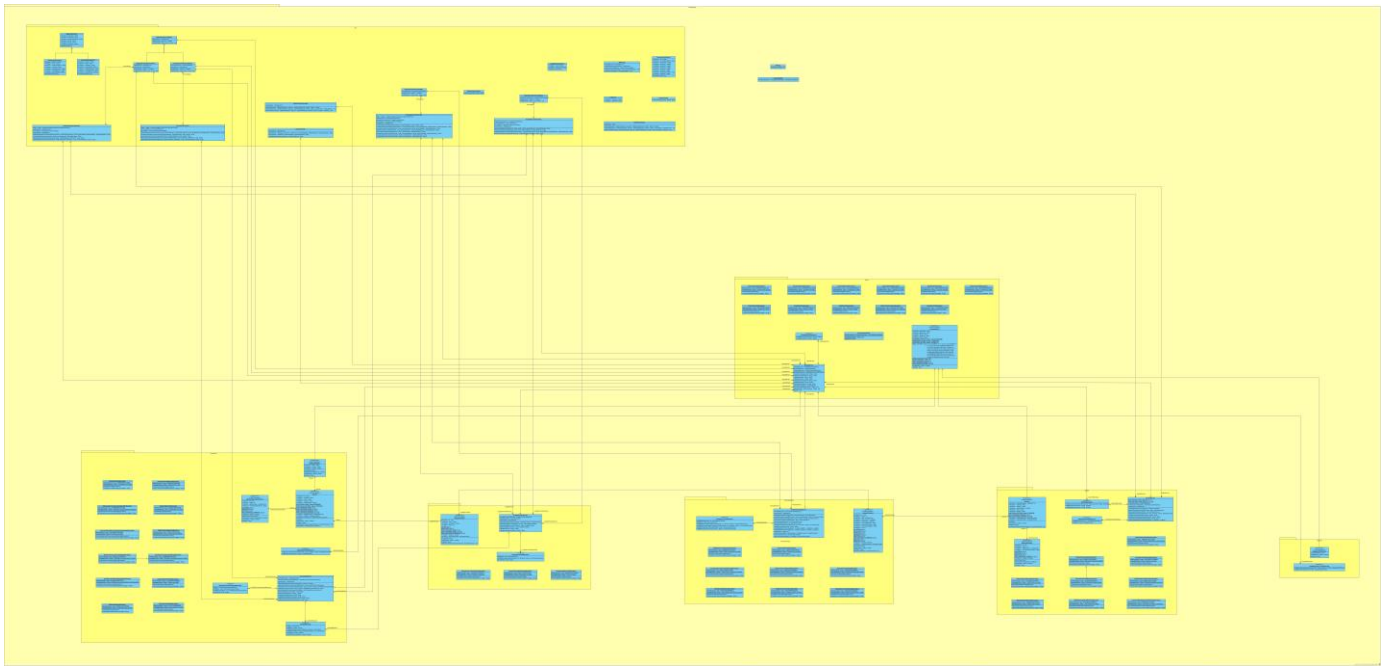
È possibile reperire la documentazione relativa all'interfaccia pubblica delle varie classi nei file Javadoc allegati presenti in /docs/javadoc/.

Tali documenti includono la definizione di precondizioni. postcondizioni e inviarianti per i metodi delle classi coinvolte.

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

4. Class diagram

Una versione a dimensioni reali del seguente diagramma è disponibile in /docs/src/diagrammi/class/.



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 tag-oriented con il back end

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