Object Design Document

Mensa Digitale

Riferimento	
Versione	2.0
Data	23/12/2020
Destinatario	Prof.ssa F. Ferrucci
Presentato da	Francesco Capriglione, Oleg Bilovus, Antonio Cacciapuoti, Antonio Giametta, Simone Masullo, Paolo Pisapia, Raffaele Squillante
Approvato da	

Revision History

Data	Versione	Descrizione	Autori
23/12/2020	0.1	Prima stesura	Tutti
14/01/2020	1.0	Stesura finale	Tutti
15/01/2020	2.0	Aggiunta di UML per Design Patterns	Simone Masullo

Sommario

Revision History 2
1. Introduzione 4
1.1. Object design trade-offs 4
1.1.1. Componenti off-the-shelf 4
1.2. Linee guida per la documentazione dell'interfaccia 6
1.3. Definizioni, acronimi e abbreviazione 7
1.4. Riferimenti 7
2. Packages 8
2.1. Divisione in pacchetti 8
2.2. Organizzazione del codice in file 12
3. Interfacce delle classi 13
4. Design Pattern con class diagram 13
5. Glossario 15

1. Introduzione

1.1. Object design trade-offs

Durante la fase di Object Design sono sorti diversi compromessi di progettazione. Di seguito sono riportati i trade-off:

- **Affidabilità vs Tempo di risposta**: Il sistema sarà implementato in modo tale da preferire l'affidabilità al tempo di risposta in quanto si garantirà un controllo accurato dei dati in input per minimizzare errori.
- **Usabilità vs Funzionalità**: Il sistema sarà facilmente usufruibile dall'utente perché verrà realizzata un'interfaccia grafica, chiara e concisa, simile a quella dell'attuale sistema Esse3, a discapito delle funzionalità offerte da quest' ultimo.
- **Leggibilità vs Tempo**: L'obiettivo sarà scrivere codice che rispetti lo standard proposto da Google per la programmazione con il linguaggio java, con l'aggiunta di commenti per eventuali chiarimenti. Questo favorirà la leggibilità ed agevolare il processo di mantenimento e modifica del codice. Tuttavia, questo vantaggio aumenterà il tempo necessario per lo sviluppo e la realizzazione dell'intero sistema.
- **Sicurezza vs Costi**: La sicurezza rappresenta uno degli aspetti principali del sistema. Tuttavia, a causa di tempi di sviluppo molto limitati, ci limiteremo ad implementare sistemi di sicurezza basati su Google Sign-In per email e password.
- **Sviluppo rapido vs Features**: Le funzionalità specifiche dell'applicazione verranno realizzate seguendo un sistema basato su delle priorità. Privilegiando uno sviluppo rapido, verrò data la precedenza agli elementi che dispongono di una priorità alta per poi integrare le restanti funzionalità in un secondo momento.

1.1.1. Componenti off-the-shelf

MensaDigitale farà affidamento su varie componenti off-the-shelf sia per il front-end che per il back-end. Ciò nasce dalla necessità di fornire un prodotto finito e funzionale in tempi stringenti, tenendo comunque in considerazione che il budget economico e le risorse umane sono limitate.

Per quanto riguarda il front-end verranno usate le librerie Jquery (v.3.5), Bootstrap (v.4.4), Google Sign-In e le API di PayPal (v.2).
JQuery è una libreria JavaScript che consentirà di scrivere buona parte della logica front-end dell'applicazione in maniera più veloce ed efficace. Sarà di particolare aiuto nell'implementazione della tecnologia AJAX e nella manipolazione del DOM. Il suo utilizzo permetterà inoltre di semplificare gran

parte della gestione delle compatibilità cross-browser, sollevando i programmatori da tale responsabilità e assicurando un corretto funzionamento del prodotto su ogni piattaforma.
Bootstrap è un framework che contiene modelli di progettazione basati su CSS e JavaScript che agevolerà la realizzazione di una parte grafica dell'applicazione che sia responsive e ben strutturata, garantendo la stessa qualità nell'esperienza di utilizzo indipendentemente dall'hardware (ad esempio le dimensioni dello schermo del device o la modalità di interazione con l'applicazione) e dal software (browser e sistema operativo). Google Sign-In è una libreria fornita da Google che si basa sul protocollo di rete OAuth per l'autorizzazione. Consente di ottenere e gestire un token per accedere a specifici dati utente conservati sulla piattaforma Google. Sarà necessaria per la gestione degli utenti di MensaDigitale.
Infine, le API di PayPal saranno necessarie per integrare PayPal in MensaDigitale quale strumento di pagamento digitale.

Lato back-end saranno invece utilizzate le seguenti componenti off-the-shelf: MySQL (v.8.0), Tomcat (v.9.X), Maven(v.4.0), Zxing(v.3.4.1).
MySQL è un celebre RDBMS di Oracle. Il suo utilizzo sarà fondamentale per la realizzazione, la gestione e l'implementazione della base di dati di MensaDigitale. Tutti i dati che per necessità di business devono essere salvati persistentemente saranno affidati ad un database basato su MySQL. Sarà inoltre utilizzata la componente MySQL Connector/J (v.8.0.23), il driver JDBC ufficiale per MySQL necessario per la comunicazione con la base di dati da Java.
Tomcat è un server web open-source sviluppato dalla Apache Software Foundation. Il suo impiego è necessario, poiché tutto il back-end di MensaDigitale (nucleo di elaborazione principale, comunicazione con la base di dati) sarà eseguito in ambiente Tomcat. Esso permetterà di utilizzare le Java Servlet per l'implementazione della logica di business e le Java Servlet Pages per la generazione dinamica delle pagine web utilizzate dal client.
Maven è uno strumento di gestione dei progetti software basati su Java, anch'esso sviluppato dalla Apache Software Foundation. Agevolerà la gestione delle varie librerie utilizzate, delle loro versioni e delle dipendenze tra esse, permettendo agli sviluppatori di concentrarsi esclusivamente sulla scrittura del codice.
Zxing è una libreria Java open-source per l'elaborazione di barcode. Sarà utilizzata per la generazione di codici QR, che fungeranno da identificativo per ciascuna prenotazione.

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.

Tipi	Regole per la denominazione	Esempi
Package	Il prefisso di un nome di pacchetto univoco sempre scritto in minuscolo e tutte le lettere ASCII minuscole tranne le abbreviazioni come UI che sarebbe User Interface in modo da non rendere il nome del pacchetto lungo.	package mensa; package pippo;
Classi	I nomi delle classi dovrebbero essere sostantivi, scritti in CamelCase. Abbiamo mantenuto i nomi delle nostre classi semplici e descrittivi. Tuttavia, abbiamo utilizzato parole intere ed evitato per quanto possibile acronimi e abbreviazioni (a meno che l'abbreviazione non sia molto più utilizzata rispetto alla forma lunga, come DB per database e UI per interfaccia utente)	class PiattoDB; class Tracciamento;
Interfacce	I nomi delle interfacce devono essere scritti in CamelCase come i nomi delle classi.	interface Identificativo;
Metodi	I nostri metodi sono verbi, scritti nella forma camelCase (da notare la prima lettera minuscola).	insert(); createReport();
Variabili	I nomi delle variabili sono brevi ma significativi. La scelta del nome di una variabile è mnemonica, cioè finalizzata a indicare all'osservatore casuale l'intento del suo utilizzo. I nomi delle variabili composte da una sola lettera vengono evitati tranne che per le variabili "usa e getta" temporanee.	float prezzo; int i;

Costanti	I nomi delle variabili dichiarate costanti di classe e delle costanti ANSI devono essere tutti maiuscoli con parole separate da trattini bassi ("_"). (Le costanti ANSI dovrebbero essere evitate, per facilitare il debug.)	static final int MIN_WIDTH = 4;

1.3. Definizioni, acronimi e abbreviazione

Termine	Definizione

Acronimo	Definizione

Abbreviazione	Definizione

1.4. Riferimenti

Bernd Bruegge, Allen H. Dutoit - Object-Oriented Software Engineering

MD_RAD_V_3

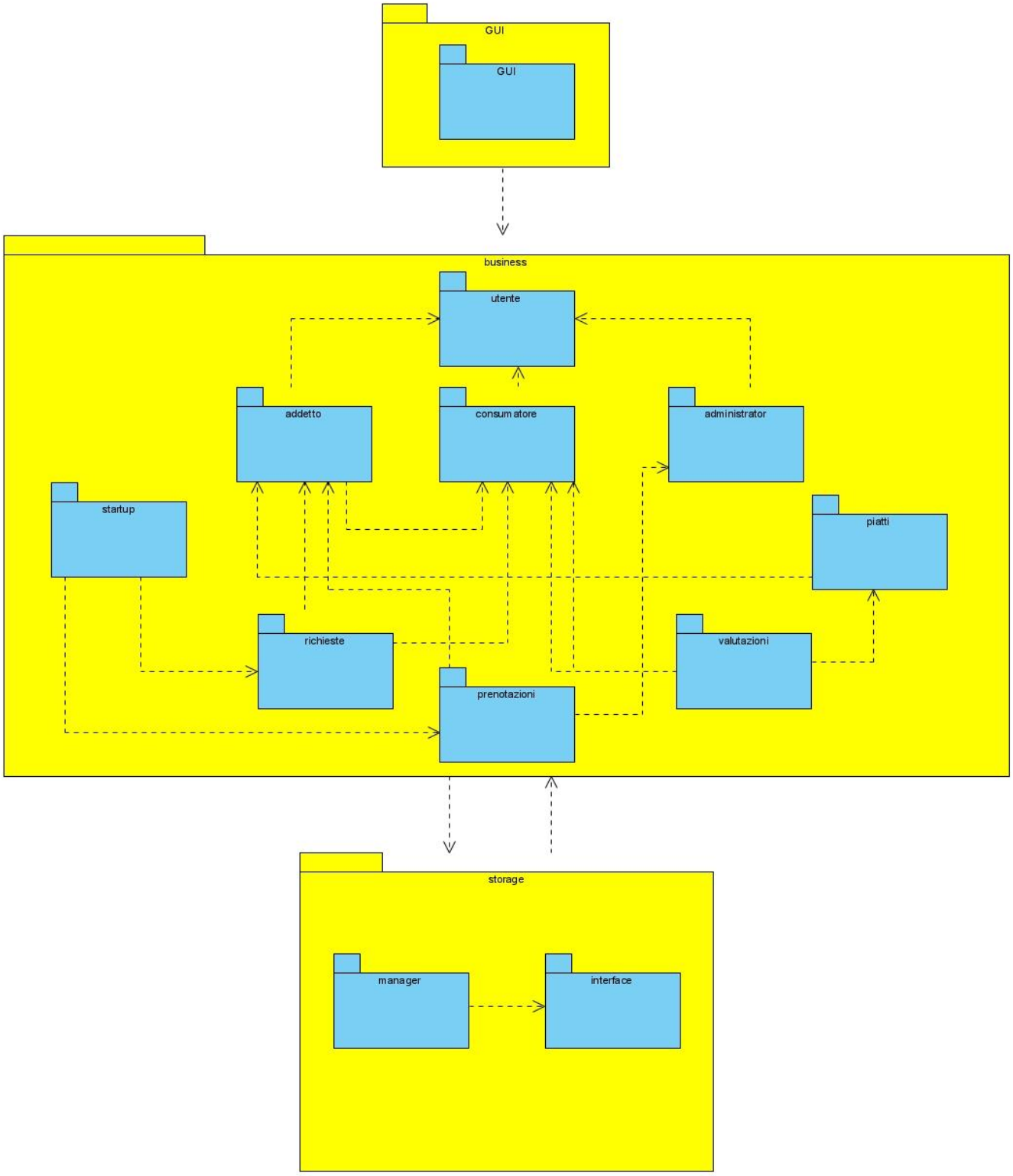
MD_SDD_V_2

2. Packages

In questa sezione viene presentato 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 rispecchia quasi in toto quella proposta nel documento di System Design: il sistema, quindi, è diviso in tre layer, ognuno dei quali contiene un discreto numero di sottosistemi e per ognuno di essi sarà realizzato un pacchetto.
Questa divisione nasce dall'esigenza di accorpare nello stesso pacchetto le classi che sono legate da una forte dipendenza da un punto di vista semantico: la divisione è stata infatti verticale in modo da ridurre le interdipendenze tra pacchetti, tuttavia presenti a causa della ripartizione degli oggetti di dominio in pacchetti diversi.
Di seguito viene riportato lo schema di decomposizione in sottosistemi e la descrizione, comprensiva di dipendenze per ognuno di pacchetti.


GUI
GUI
business
utente
addetto
consumatore
administrator
startup
piatti
richieste
valutazioni
prenotazioni
storage
manager
interface

Per una migliore comprensione, si evidenzia che il pacchetto relativo alla GUI dipende dal package Business. E che quest'ultimo dipende reciprocamente con il pacchetto Storage.

All'interno del package GUI tutti i file che implementano l'interfaccia grafica sono raggruppati in un unico package.

Nome Pacchetto	GUI
Descrizione	Definisce le proprietà necessarie affinché venga mostrata l'interfaccia grafica all'utente
Dipendenze	Business

Nome Pacchetto	business.utente
Descrizione	Definisce le proprietà dell'utente generico della piattaforma ed espone i servizi di autenticazione, logout, e ottenimento dell'autenticazione
Dipendenze	storage.interface

Nome Pacchetto	business.admin
Descrizione	Definisce le proprietà di admin.
Dipendenze	storage.interface business.utente

Nome Pacchetto	business.consumatore
Descrizione	Definisce le proprietà dei consumatori generici, ed espone i servizi di visualizzazione delle informazioni sugli stessi
Dipendenze	storage.interface business.utente

Nome Pacchetto	business.addetto
Descrizione	Definisce le proprietà degli addetti generici, ed espone i servizi di visualizzazione delle informazioni sugli stessi
Dipendenze	storage.interface business.utente

Nome Pacchetto	business.prenotazioni
Descrizione	Definisce le proprietà delle prenotazioni generiche e delle fasce orarie disponibili nel sistema.
Dipendenze	storage.interface business.consumatore business.addetto business.administrator

Nome Pacchetto	business.piatti
Descrizione	Definisce le proprietà dei piatti generici, ed espone i servizi di visualizzazione degli stessi
Dipendenze	storage.interface business.addetto

Nome Pacchetto	business.richieste
Descrizione	Definisce le proprietà delle richieste d'attivazione dei servizi di ristorazione generiche, ed espone i servizi di visualizzazione delle stesse
Dipendenze	btorage.interface business.consumatore business.addetto

Nome Pacchetto	business.valutazioni
Descrizione	Definisce le proprietà delle valutazioni dei piatti generiche, ed espone i servizi di visualizzazione delle stesse
Dipendenze	storage.interface

Nome Pacchetto	storage.manager
Descrizione	Definisce l'implementazione delle interfacce per la comunicazione con la base dati
Dipendenze	storage.interface

Nome Pacchetto	storage.interface
Descrizione	Definisce le operazioni da poter eseguire sulla base dati
Dipendenze	business

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.mensadigitale(e saranno mappati nel rispettivo percorso src/main/java/…) ad eccezione del pacchetto realizzante l'interfaccia utente, che sarà invece collocato nella directory src/main/webapp/.

3. Interfacce delle classi

La documentazione relativa all'interfaccia pubblica delle classi è reperibile nei vari file Javadoc presenti in docs/javadoc/.

Tali documenti presentano pre-condizioni, post-condizioni, definizione degli eventuali parametri in input dei metodi e descrizione di eventuali parametri output oltre ad una breve spiegazione sul funzionamento dei metodi ove necessaria.

4. Design Pattern con class diagram

Design patterns

Nello sviluppo del sistema abbiamo deciso di utilizzare dei design patterns per evitare di rielaborare soluzioni già precedentemente fornite da altri programmatori.

I design patterns utilizzati sono:

- Adapter per risolvere i problemi di compatibilità tra il sistema ed il database
- Singleton per garantire l'esistenza di un'unica istanza di determinate classi
- ObjectPool per garantire velocità nella generazione di connessioni al database

- **Adapter**

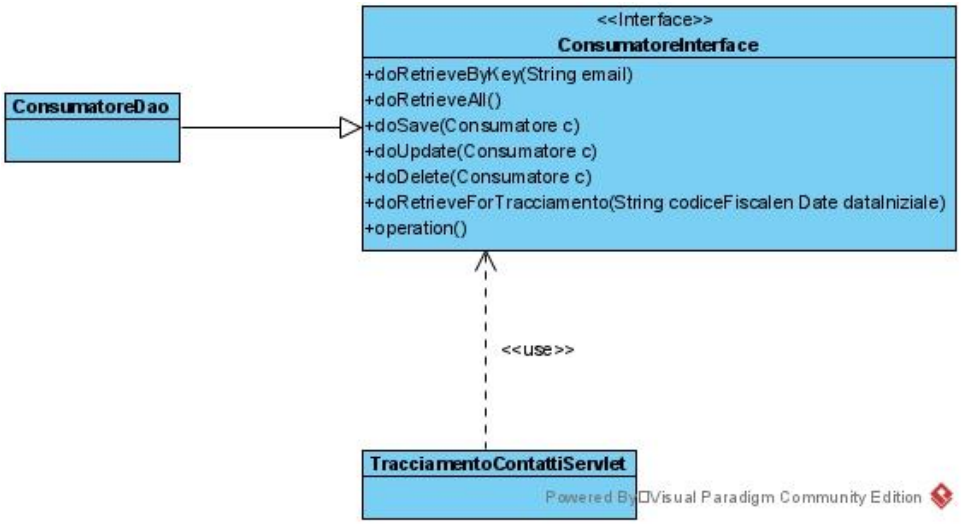
  Nome del design pattern: Adapter

  Descrizione del problema: vogliamo poter scrivere codice che non necessiterà di grandi modifiche se un giorno dovesse cambiare il DBMS.

  Soluzione del problema: sviluppiamo un'interfaccia basandoci non tanto sulle istruzioni che il Database si aspetta, quanto sulle operazioni che l'oggetto cliente vuole effettuare.

  Conseguenze: il Client utilizza lo stesso codice per accedere al Database a prescindere dalla scelta del DBMS poiché viene aggiunto un livello di astrazione con conseguente riduzione del livello di accoppiamento tra DBMS ed i sottosistemi che vogliono accedervi.

  Il codice risulta più manutenibile in quanto in caso di cambiamento di DBMS sarà necessario apportare modifiche unicamente all'Adapter.



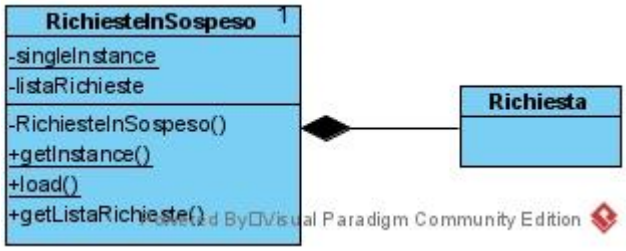
- **Singleton**

  Nome del design pattern: Singleton

  Descrizione del problema: vogliamo assicurarci che esista un'unica istanza della classe RichiesteInSospeso affinché tale risorsa possa essere correttamente condivisa.

  Soluzione del problema: utilizziamo il design pattern Singleton fornendo alla classe RichiesteInSospeso un costruttore privato. Ciò ci assicura che non verranno create più istanze di tale classe e che le richieste inoltrate dagli studenti siano sempre correttamente memorizzate.

  Conseguenze: è assicurato il corretto accesso alla classe che si occupa di memorizzare le richieste in sospeso, gli aggiornamenti dell'insieme delle richieste in sospeso non richiedono alcun accesso al Database se non per memorizzare la richiesta dello studente ma l'utilizzo di un Singleton comporta la gestione della concorrenza in caso di scrittura.



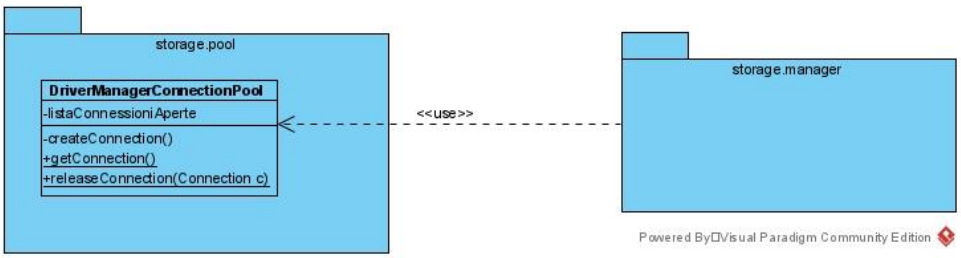
- **Object Pool**

  Nome del design pattern: Object Pool

  Descrizione del problema: siccome ci aspettiamo più richieste da diversi utenti non possiamo aspettarci che il nostro sistema sia in grado di aprire nuove connessioni per ogni utente e rispondere alle richieste in un tempo accettabile.

  Soluzione del problema: ogni volta che un utente effettua una richiesta che richiede una connessione al database il sistema controlla che ci siano connessioni già aperte in un pool di connessioni. Se ciò è vero, il sistema restituisce una connessione già aperta, altrimenti ne apre una nuova. Quando la connessione viene rilasciata essa torna nel pool delle connessioni aperte, in attesa di essere utilizzata.

  Conseguenze: il sistema è in grado di rispondere alle richieste più velocemente poiché non ha sempre bisogno di aprire nuove connessioni ma potrebbe trovarne di già aperte.



Il class diagram generale è reperibile al seguente indirizzo:
https://ob-unisa.github.io/MensaDigitale/ClassDiagram.svg

5. Glossario

Termine	Definizione