Safe
Meeting

ODD
OBJECT
DESIGN
DOCUMENT

Safe Meeting

ODD 3.0 – ultima modifica 07/01/2019

Destinatario del documento: Top Manager – Prof.ssa F. Ferrucci

Presentato da:

Alfonso Serio	Emilio Mainardi	Gianluca Verlingieri	Luca Di Chiara
Alessandra Capo	Donato Marmora	Andrea Califano	Matteo Ferrara

Approvato da:

Roberta Gesumaria	Francesca Tassatone

Storia dei cambiamenti

Data	Versione	Cambiamenti	Autore
09/12/2018	ODD 0.1	Strutturazione documento ed inserimento dei trade off.	Tutti
10/12/2018	ODD 0.2	Inserimento dei componenti off the shelfs, definizione design pattern, linee guida per l'implementazione e definizioni, acronimi abbreviazioni e riferimenti.	Matteo Ferrara, Alfonso Serio, Luca Di Chiara
11/12/2018	ODD 0.3	Inserimento dei packages e interfacce delle classi.	Gianluca Verlingieri, Donato Marmora, Matteo Ferrara.
12/12/2018	ODD 1.0	Revisione ODD.	Gianluca Verlingieri
15/12/2018	ODD 2.0	Modifica packages e interfaccia delle classi.	Gianluca Verlingieri, Donato Marmora
07/01/2019	ODD 3.0	Modificate le interfacce delle classi e i packages dopo l'implementazione.	Gianluca Verlingieri, Donato Marmora

Sommario

1 – Introduzione

Dopo la realizzazione dei documenti RAD e SDD abbiamo descritto in linea di massima quello che sarà il nostro sistema e quindi i nostri obiettivi, tralasciando gli aspetti implementativi. Il seguente documento ha lo scopo di produrre un modello capace di integrare in modo coerente e preciso tutte le funzionalità individuate nelle fasi precedenti. In particolare, definisce 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.

1.1 - Trade-offs

Comprensibilità Vs interfaccia:
Il codice deve essere quanto più comprensibile possibile per facilitare la fase di testing ed eventuali future modifiche. Il codice sarà quindi accompagnato da commenti che ne semplifichino la comprensione. Ovviamente questa caratteristica aggiungerà un incremento di tempo allo sviluppo del nostro progetto.

Sicurezza Vs Tempi di risposta:
Essendo la sicurezza uno degli aspetti importanti del sistema si è preferito l'utilizzo di un algoritmo di cifratura a discapito dei tempi di risposta in quanto il sistema può impiegare più tempo per l'accesso al database.

Tempo di risposta Vs Memoria:
Per gestire i dati della piattaforma si fa uso di un database relazionale in quanto è un sistema che garantisce tempi di risposta veloci e accesso concorrente ai dati. Utilizzare un database implica però un overhead necessario per la gestione che incide prevalentemente sullo spazio maggiore che occorre rispetto ad altre metodologie di mantenimento della persistenza dei dati.

Costo Vs Mantenimento:
Per minimizzare i costi si è deciso di utilizzare risorse open source come Bootstrap.

1.2 – Componenti off-the-shelf

Per contenere gli eventuali costi di manutenzione e, allo stesso tempo, fruire di un supporto già pronto all'uso ed open source, ci avverremo di Bootstrap.
In tal modo non sarà necessario impiegare tempo per le molteplici configurazioni ma sarà possibile implementare facilmente tutti gli elementi caratteristici di un front end e tra gli altri:

- Pulsanti;
- Elementi di navigazione;
- Thumbnail;
- Indicazioni degli errori.

Per ultimo, ma non per importanza, un beneficio ottenuto da questa scelta riguarda il Responsive Design o design adattivo; il quale consentirà di realizzare pagine Web capaci di adattarsi dinamicamente a qualsiasi dispositivo.

1.3 – Linee guida per l'implementazione dell'interfaccia

Standard di codifica

Nella fase di implementazione del sistema, gli sviluppatori dovranno attenersi alle seguenti linee guida.

Pagine HTML

Indentazione

L'indentazione deve essere effettuata con un TAB, i tag devono essere opportunamente indentati.

Es.

```
<html>
<head>
<title>esempio</title>
</head>
<body>
</body>
</html>
```

Deve essere sostituito da:

```
<html>
    <head>
        <title>esempio</title>
    </head>
    <body>
    </body>
</html>
```

Chiusura tag

Ogni tag di apertura deve essere necessariamente seguito dall'apposito tag di chiusura (eccetto i tag self- closing).

Contenuto diviso dallo stile

Il codice HTML non deve essere mischiato al codice CSS, ma lo stile deve essere opportunamente disgiunto dal contenuto della pagina web.

Fogli di stile CSS

I fogli di stile devono essere raggruppati in opportune cartelle. Lo stile non deve essere unito con il contenuto della pagina, scritto in HTML.

Script javascript e JSP Indentazione

Indentare opportunamente le istruzioni attraverso l'utilizzo di un TAB.

Commenti

Ogni metodo e ogni file devono essere preceduti da un commento che riporti l'obiettivo che si vuole e deve raggiungere con il nome dell'autore.

Nomi

La notazione da usare per assegnare nomi alle variabili e ai metodi è la nota Camel Notation. I nomi dei file, delle operazioni e delle variabili devono essere evocativi e in lingua italiana. Le uniche eccezioni sono GET e SET che sono stati inseriti nella nostra terminologia data la familiarità con il linguaggio JAVA e soprattutto perché anche la migliore traduzione in italiano non regge il confronto.

Posizione dichiarazione variabili

Posizionare le dichiarazioni all'inizio dei blocchi. Non bisogna dichiarare le variabili al loro primo uso: può confondere il programmatore inesperto e impedire la portabilità del codice dentro lo scope. L'unica eccezione a questa regola sono gli indici dei cicli for che possono essere dichiarati a più alto livello. Ad esempio, non dichiarare una variabile con lo stesso nome in un blocco interno.

Parentesi

A prescindere dalle istruzioni che seguono un IF, è necessario riportare il blocco di istruzioni tra parentesi graffe.

Nessun valore hard coded

Una convenzione importante, per quanto riguarda l'inserimento di numeri o di valori costanti, è quella di non usare una codifica fissa (hard coded), che è fortemente sconsigliata, ma di associare sempre il valore ad una variabile o semplicemente definire una macro che può essere richiamata da eventi ed essere parametrizzata. In questo modo si facilita la modifica, sostituendo solo il valore della variabile o macro, in un unico posto.

Database SQL

I nomi delle tabelle e dei campi devono essere: evocativi, leggibili.

Standard per l'organizzazione dei file

Ogni file deve essere:

• Sviluppato e diviso in base alla categoria di appartenenza, ovvero deve essere correlato ad un'unica funzionalità che persegue. Ogni pagina deve essere implementata in file separati;

• Diviso in più file se raggiunge una lunghezza tale da divenire difficile da leggere e comprendere.

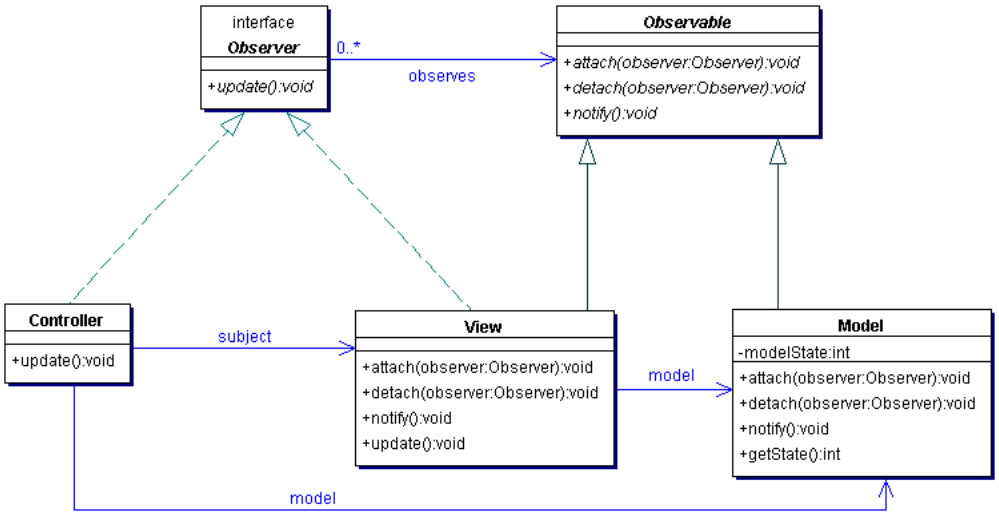
Organizzare in una cartella i file della libreria usate e le altre risorse scaricate necessarie per lo sviluppo del progetto.

1.4 – Design Pattern

Observer Pattern

Nato dall'esigenza di mantenere un alto livello di consistenza fra classi correlate ed utilizzato come base architetturale di molti sistemi di gestione di eventi, l'observer pattern si basa su uno o più oggetti, chiamati osservatori o observer, che vengono registrati per gestire un evento che potrebbe essere generato dall'oggetto "osservato" (che può essere chiamato soggetto).

Tale pattern ci consentirà quindi di definire una dipendenza uno a molti fra oggetti. Nello specifico, se un oggetto cambia il suo stato interno, ciascuno degli oggetti dipendenti da esso viene notificato e aggiornato automaticamente.



Ecco i principali benefici dati dal suo utilizzo:

- Consente di inviare dati ad altri oggetti in modo efficace senza alcuna modifica nelle classi Subject o Observer.
- I Subject e Observer sono indipendenti. Una modifica di uno di questi componenti non richiede la modifica dell'altro componente.
- Gli Observers possono essere aggiunti o rimossi in qualsiasi momento.

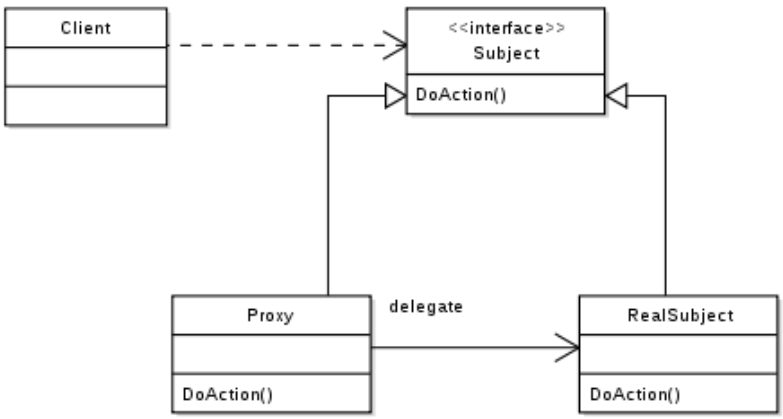
Questo pattern ci tornerà utile nella gestione e nella comunicazione dell'assenza da parte del docente. Una volta comunicata l'assenza del docente in studio ed inviato un messaggio a tutti gli studenti prenotati per un eventuale ricevimento, verranno automaticamente aggiornate le liste degli studenti prenotati in quell'arco temporale.

Proxy **Pattern**

Semplice ma allo stesso tempo molto utile, il Proxy Pattern nasce principalmente per risolvere problematiche legate all'accesso ad un oggetto che richiede tempi importanti per la creazione o per essere raggiunto.

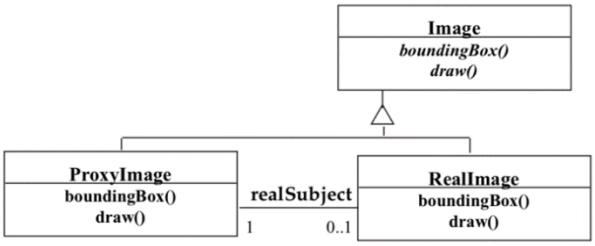
Tale pattern prevede quindi la creazione di un oggetto "proxy" che viene usato al posto dell'oggetto reale e che quindi deve avere necessariamente la stessa "forma" dell'oggetto che sostituisce. Insomma, crea una sorta di un "surrogato" (o segnaposto) per un altro oggetto di cui si desidera controllare l'accesso.
Ecco i principali benefici dati dal suo utilizzo:

- Distanza tra "Concetto" ed Implementazione minimizzata.
- Maggiore portabilità del codice del client. Le modifiche sono incapsulate nell'oggetto Proxy.



Avremo quindi senza dubbio vantaggio in termini di tempi di sviluppo e di eventuale modifica degli oggetti durante la programmazione, e al tempo stesso vi saranno vantaggi per gli utenti che utilizzeranno la piattaforma; dato che questo pattern consente di posticipare l'effettivo accesso alle risorse dell'oggetto quando è davvero necessario, consentendo maggiore fluidità all'elaborazione.

Definito "proxy di sincronizzazione", questo pattern ci sarà ancora utile in quanto capace di regolare l'accesso ad un oggetto sottoposto a più richieste. Infine, ci avvaremo soprattutto del "proxy remoto", capace di avere l'accesso a risorse distribuite sulla rete come se fossero accessibili come oggetto locale (è il meccanismo Remote Methode Invocation di Java). Nello specifico, verranno gestite le foto dei docenti sulla piattaforma.



Le immagini verranno memorizzate e caricate separatamente dal testo. Se non viene caricata una "*RealImage*" (immagine reale), una "*ProxyImage*" visualizza un rettangolo grigio in luogo dell'immagine. Il client non può distinguere che si tratta di un "*ProxyImage*" invece di una "*RealImage"*.

1.5 – Definizioni, acronimi e abbreviazioni

Studio: rappresenta lo studio di un docente, nonché luogo di incontro tra docenti e studenti.

Studente: rappresenta uno studente universitario, ognuno avente un proprio account in cui saranno specificate le seguenti informazioni:

1. **Nome;**
2. **Cognome;**
3. **E-mail istituzionale;**
4. **Matricola;**
5. **Password.**

Docente: rappresenta un docente universitario, ognuno avente un proprio account in cui saranno specificate le seguenti informazioni:

1. **Nome;**
2. **Cognome;**
3. **E-mail istituzionale;**
4. **Corsi;**
5. **Calendario ricevimento;**
6. **Password.**

Corso: rappresenta un corso universitario tenuto da uno o più docenti.

Prenotazione: rappresenta una prenotazione di ricevimento effettuata da uno studente.

Ricevimento: rappresenta l'accoglienza di uno studente da parte di un docente dopo aver effettuato la prenotazione.

Acronimi

SM = Safe Meeting
RAD = Requirement Analysis Document.
SDD = System Design Document.
ODD = Object Design Document.
RF = Requisito Funzionale.
RNF = Requisito Non Funzionale.
SC = Scenario.
UC = Use Case.
UCD = Use Case Diagram.
CD = Class Diagram.
SD = Sequence Diagram.
SCD = Statechart Diagram.
NP = Navigation Path.
MU = Mock-up.
MVC = Model View Controller.
SQL = Structured Query Language.
FURPS+ = Rappresenta:

- Funzionalità;
- Usabilità;
- Affidabilità;
- Prestazioni;
- Supportabilità.

Il "+" sta per pseudo-requisiti o vincoli del sistema:

- Implementazione;
- Interfaccia;
- Operazioni;
- Packaging;
- Legali.

AUT = Autenticazione.
ACC = Account.
STUD = Studente.
DOC = Docente.
NA = Nessuna.

1.6 – Riferimenti

- http://www.safemeeting.it
- Bernd Bruegge & Allen H. Dutoit, Object-Oriented Software Engineering: Using UML, Patterns and Java, (2nd edition), Prentice-Hall, 2004.

- Ian Sommerville, Ingegneria del software, (8a edizione), Pearson, 2007.
- RAD_SM_v3.0.
- SDD_SM_v1.0

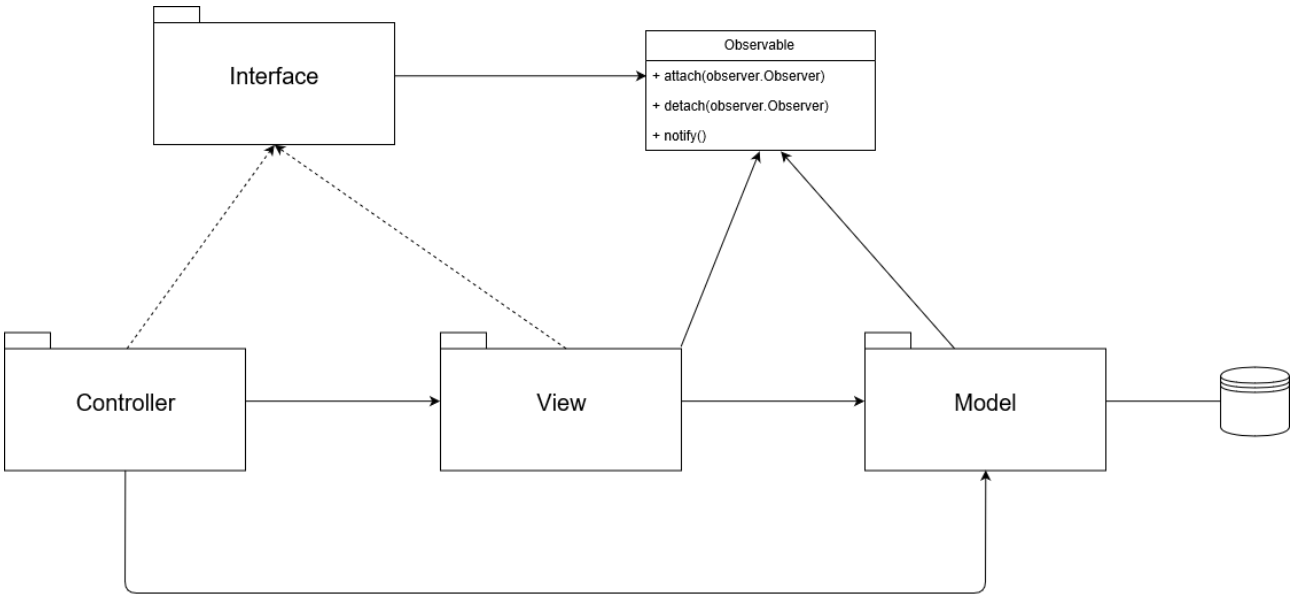
2 – Packages

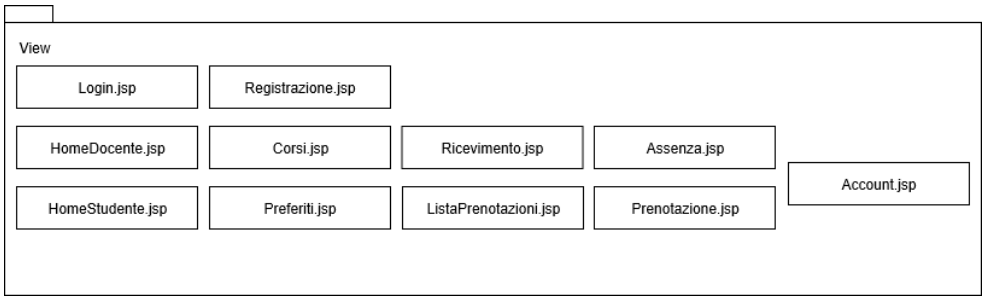
In questo paragrafo è riportata la decomposizione dei sottosistemi nei vari packages delle classi Java. Abbiamo incluso i design pattern "Observer" e "Proxy".

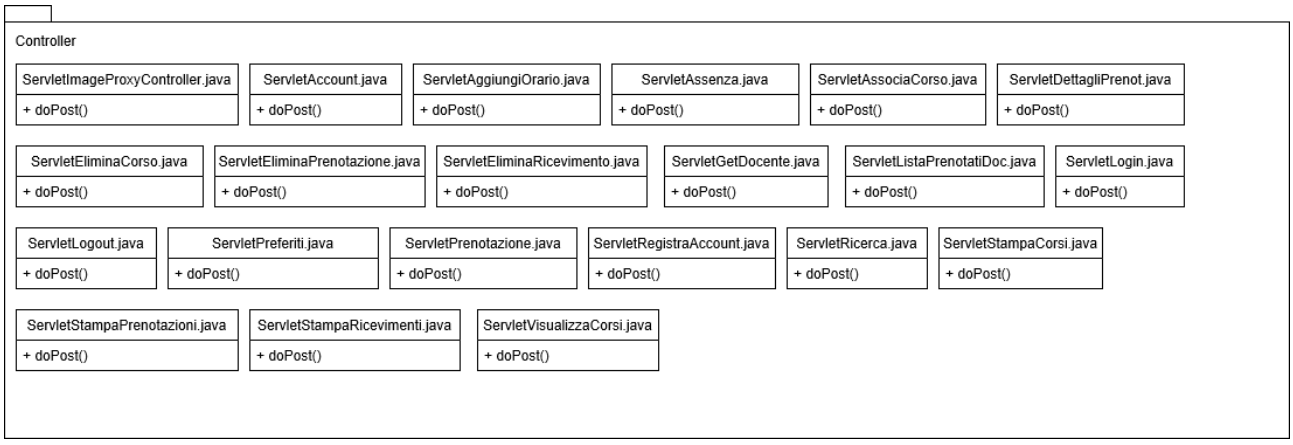
I package derivanti sono:

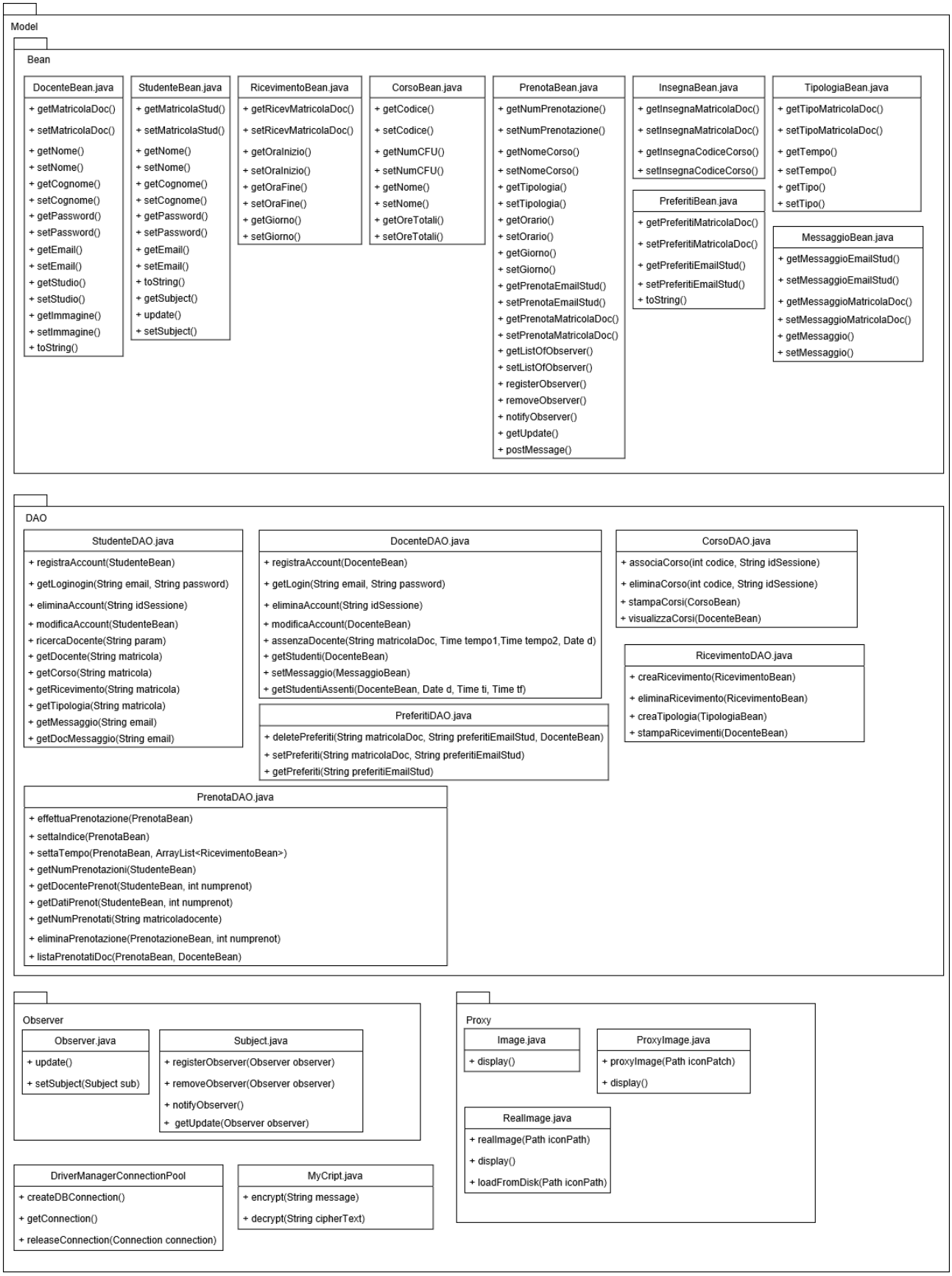
- safemeeting.view, che contiene tutte le pagine .jsp create dinamicamente;
- safemeeting.controller, che contiene tutte le servlet usate per gestire il collegamento tra le pagine e gli elementi sottostanti usati per gestire la logica del sistema.
- safemeeting.interface, che contiene la classe observer, usata per applicare il pattern Observer;
- safemeeting.model, contiene due sotto-packages, rappresentati nel modello per ordine logico ma effettivamente implementati come un unico package Model a causa di limitazioni di Eclipse, che sono:
    - safemeeting.model.bean, contiene tutti i bean che servono per creare gli oggetti utilizzati per l'implementazione.
    - safemeeting.model.DAO, contiene tutti i DAO che servono per interfacciarci con il DBMS relazionale ed interrogarlo tramite dei metodi specifici.
    - safemeeting.model.DriveManagerConnectionPool, classe utilizzata per le connessioni al database.
    - safemeeting.model.MyCript, classe utilizzata per la crittografia delle password.
    - safemeeting.model.observer, contiene le classi utili per far funzionare l'Observer Pattern.
    - safemeeting.model.proxy, contiene le classi utili per far funzionare il Proxy Pattern.

Inoltre, è esposta la classe "Observable" che non è associabile ad un package preciso, ma la sua presenza è indispensabile per il corretto funzionamento dell'Observer patten citato in precedenza. Safemeetingview in fase di implementazione diventerà Web Content.


Interface
Observable
+ attach(observer.Observer)
+ detach(observer.Observer)
+ notify()
Controller
View
Model


View
Login.jsp
Registrazione.jsp
HomeDocente.jsp
Corsi.jsp
Ricevimento.jsp
Assenza.jsp
Account.jsp
HomeStudente.jsp
Preferiti.jsp
ListaPrenotazioni.jsp
Prenotazione.jsp


Controller
ServletImageProxyController.java
+ doPost()
ServletAccount.java
+ doPost()
ServletAggiungiOrario.java
+ doPost()
ServletAssenza.java
+ doPost()
ServletAssociaCorso.java
+ doPost()
ServletDettagliPrenot.java
+ doPost()
ServletEliminaCorso.java
+ doPost()
ServletEliminaPrenotazione.java
+ doPost()
ServletEliminaRicevimento.java
+ doPost()
ServletGetDocente.java
+ doPost()
ServletListaPrenotatiDoc.java
+ doPost()
ServletLogin.java
+ doPost()
ServletLogout.java
+ doPost()
ServletPreferiti.java
+ doPost()
ServletPrenotazione.java
+ doPost()
ServletRegistraAccount.java
+ doPost()
ServletRicerca.java
+ doPost()
ServletStampaCorsi.java
+ doPost()
ServletStampaPrenotazioni.java
+ doPost()
ServletStampaRicevimenti.java
+ doPost()
ServletVisualizzaCorsi.java
+ doPost()


Model
Bean
DocenteBean.java
+ getMatricolaDoc()
+ setMatricolaDoc()
+ getNome()
+ setNome()
+ getCognome()
+ setCognome()
+ getPassword()
+ setPassword()
+ getEmail()
+ setEmail()
+ getStudio()
+ setStudio()
+ getImmagine()
+ setImmagine()
+ toString()
StudenteBean.java
+ getMatricolaStud()
+ setMatricolaStud()
+ getNome()
+ setNome()
+ getCognome()
+ setCognome()
+ getPassword()
+ setPassword()
+ getEmail()
+ setEmail()
+ toString()
+ getSubject()
+ update()
+ setSubject()
RicevimentoBean.java
+ getRicevMatricolaDoc()
+ setRicevMatricolaDoc()
+ getOraInizio()
+ setOraInizio()
+ getOraFine()
+ setOraFine()
+ getGiorno()
+ setGiorno()
CorsoBean.java
+ getCodice()
+ setCodice()
+ getNumCFU()
+ setNumCFU()
+ getNome()
+ setNome()
+ getOreTotali()
+ setOreTotali()
PrenotaBean.java
+ getNumPrenotazione()
+ setNumPrenotazione()
+ getNomeCorso()
+ setNomeCorso()
+ getTipologia()
+ setTipologia()
+ getOrario()
+ setOrario()
+ getGiorno()
+ setGiorno()
+ getPrenotaEmailStud()
+ setPrenotaEmailStud()
+ getPrenotaMatricolaDoc()
+ setPrenotaMatricolaDoc()
+ getListOfObserver()
+ setListOfObserver()
+ registerObserver()
+ removeObserver()
+ notifyObserver()
+ getUpdate()
+ postMessage()
InsegnaBean.java
+ getInsegnaMatricolaDoc()
+ setInsegnaMatricolaDoc()
+ getInsegnaCodiceCorso()
+ setInsegnaCodiceCorso()
PreferitiBean.java
+ getPreferitiMatricolaDoc()
+ setPreferitiMatricolaDoc()
+ getPreferitiEmailStud()
+ setPreferitiEmailStud()
+ toString()
TipologiaBean.java
+ getTipoMatricolaDoc()
+ setTipoMatricolaDoc()
+ getTempo()
+ setTempo()
+ getTipo()
+ setTipo()
MessaggioBean.java
+ getMessaggioEmailStud()
+ setMessaggioEmailStud()
+ getMessaggioMatricolaDoc()
+ setMessaggioMatricolaDoc()
+ getMessaggio()
+ setMessaggio()
DAO
StudenteDAO.java
+ registraAccount(StudenteBean)
+ getLoginogin(String email, String password)
+ eliminaAccount(String idSessione)
+ modificaAccount(StudenteBean)
+ ricercaDocente(String param)
+ getDocente(String matricola)
+ getCorso(String matricola)
+ getRicevimento(String matricola)
+ getTipologia(String matricola)
+ getMessaggio(String email)
+ getDocMessaggio(String email)
DocenteDAO.java
+ registraAccount(DocenteBean)
+ getLogin(String email, String password)
+ eliminaAccount(String idSessione)
+ modificaAccount(DocenteBean)
+ assenzaDocente(String matricolaDoc, Time tempo1,Time tempo2, Date d)
+ getStudenti(DocenteBean)
+ setMessaggio(MessaggioBean)
+ getStudentiAssenti(DocenteBean, Date d, Time ti, Time tf)
CorsoDAO.java
+ associaCorso(int codice, String idSessione)
+ eliminaCorso(int codice, String idSessione)
+ stampaCorsi(CorsoBean)
+ visualizzaCorsi(DocenteBean)
RicevimentoDAO.java
+ creaRicevimento(RicevimentoBean)
+ eliminaRicevimento(RicevimentoBean)
+ creaTipologia(TipologiaBean)
+ stampaRicevimenti(DocenteBean)
PreferitiDAO.java
+ deletePreferiti(String matricolaDoc, String preferitiEmailStud, DocenteBean)
+ setPreferiti(String matricolaDoc, String preferitiEmailStud)
+ getPreferiti(String preferitiEmailStud)
PrenotaDAO.java
+ effettuaPrenotazione(PrenotaBean)
+ settaIndice(PrenotaBean)
+ settaTempo(PrenotaBean, ArrayList<RicevimentoBean>)
+ getNumPrenotazioni(StudenteBean)
+ getDocentePrenot(StudenteBean, int numprenot)
+ getDatiPrenot(StudenteBean, int numprenot)
+ getNumPrenotati(String matricoladocente)
+ eliminaPrenotazione(PrenotazioneBean, int numprenot)
+ listaPrenotatiDoc(PrenotaBean, DocenteBean)
Observer
Observer.java
+ update()
+ setSubject(Subject sub)
Subject.java
+ registerObserver(Observer observer)
+ removeObserver(Observer observer)
+ notifyObserver()
+ getUpdate(Observer observer)
Proxy
Image.java
+ display()
ProxyImage.java
+ proxyImage(Path iconPatch)
+ display()
RealImage.java
+ realImage(Path iconPath)
+ display()
+ loadFromDisk(Path iconPath)
DriverManagerConnectionPool
+ createDBConnection()
+ getConnection()
+ releaseConnection(Connection connection)
MyCript.java
+ encrypt(String message)
+ decrypt(String cipherText)

3 –Interfacce delle classi

Nome classe	StudenteDAO
Descrizione	Questa classe rappresenta l'entità tabellare Studente contenuta all'interno del nostro DBMS relazionale. Ci permette di interfacciarci con il DBMS relazionale e di interrogarlo tramite dei metodi specifici.
Pre-condizione	**context** StudenteDAO:: public registraAccount(StudenteBean); **pre** String nome!= null && String cognome!=null && String matricolaStud!=null && String password!=null && String emailStud!=null **context** StudenteDAO:: public getLogin(String email, String password); **pre** String emailStud!=null && String password!=null **context** StudenteDAO:: public eliminaAccount(String email); **pre** String emailStud!=null **context** StudenteDAO:: public modificaAccount(String nome, String cognome, String matricola, String oldPassword, String newPassword, String email); **pre** String nome!=null && String cognome!=null && String matricolaStud!=null && String oldPassword!=null && String newPassword!=null && String emailStud!= null **context** StudenteDAO:: public ricercaDocente(String param, DocenteBean); **pre** String param!=null **context** StudenteDAO:: public getDocente(String matricola); **pre** String matricolaDoc!=null **context** StudenteDAO:: public getCorso(String matricola); **pre** String matricolaDoc!=null **context** StudenteDAO:: public getRicevimento(String matricola); **pre** String matricolaDoc!=null **context** StudenteDAO:: public getTipologia(String matricola); **pre** String matricolaDoc!=null **context** StudenteDAO:: public getMessaggio(String email); **pre** String EmailStud!=null **context** StudenteDAO:: public getDocMessaggio(String email);

	pre String EmailStud!=null
Post-condizione	**context** StudenteDAO:: public getLogin(String email, String password) **post** getLogin= true **context** StudenteDAO:: public ricercaDocente(String param, DocenteBean); **post** ricercaDocente= true **context** StudenteDAO:: public getDocente(String matricola); **post** getDocente= true **context** StudenteDAO:: public getCorso(String matricola); **post** getCorso= true **context** StudenteDAO:: public getRicevimento(String matricola); **post** getRicevimento= true; **context** StudenteDAO:: public getTipologia(String matricola); **post** getTipologia= true
Invarianti	

Nome classe	DocenteDAO
Descrizione	Questa classe rappresenta l'entità tabellare "Docente" contenuta all'interno del nostro DBMS relazionale. Ci permette di interfacciarci con il DBMS relazionale e di interrogarlo tramite dei metodi specifici.
Pre-condizione	**context** DocenteDAO:: public registraAccount(DocenteBean); **pre** String nome!= null && String cognome!=null && String matricolaDoc!=null && String password!=null && String emailDoc!=null && String studio!= null **context** DocenteDAO:: public getLogin(String email, String password); **pre** String emailDoc!=null && String password!=null **context** DocenteDAO:: public modificaAccount(String nome, String cognome, String matricola, String studio, String oldPassword, String newPassword, String immagine) **pre** String nome!=null && String cognome!=null && String matricolaDoc!=null && String oldPassword!=null && String newPassword!=null && String immagine!= null && String studio!=null **context** DocenteDAO:: public eliminaAccount(String matricola); **pre** String matricolaDoc!=null **context** DocenteDAO:: public assenzaDocente(String matricolaDoc, Time tempo1, Time tempo2, Date d); **pre** String matricolaDoc!=null && Time ora_inizio!=null && Time ora_fine!=null && Date giorno!=null **context** StudenteDAO:: public getStudenti(DocenteBean); **pre** String matricolaDoc!=null **context** StudenteDAO:: public setMessaggio(MessaggioBean); **pre** String emailStud!=null && String matricolaDoc!=null && String messaggio!=null **context** StudenteDAO:: public getStudentiAssenza(DocenteBean, Date d, Time ti, Time tg); **pre** String matricolaDoc!=null && Date giorno!=null && Time ora_inizio!=null && Time ora_fine1=null
Post-condizione	**context** DocenteDAO:: public getLogin(String email, String password); **post** getLogin= true **context** DocenteDAO:: public getStudenti(DocenteBean);

	post getStudenti= true **context** DocenteDAO:: public getStudentiAssenza(DocenteBean, Date d, Time ti, Time tf); **post** getStudentiAssenza= true
Invarianti	

Nome classe	RicevimentoDAO
Descrizione	Questa classe rappresenta l'entità tabellare "Ricevimento" contenuta all'interno del nostro DBMS relazionale. Ci permette di interfacciarci con il DBMS relazionale e di interrogarlo tramite dei metodi specifici.
Pre- condizione	**context** RicevimentoDAO:: public creaRicevimento(Ricevimento Bean); **pre** Time ora_inizio!=null && Time ora_fine!=null && String giorno!=null && String matricolaDoc!=null **context** RicevimentoDAO:: public eliminaRicevimento(Ricevimento Bean); **pre** String matricolaDoc!=null && Time ora_inizio!=null && Time ora_fine!=null && String giorno!=null **context** RicevimentoDAO:: public creaTipologia(TipologiaBean); **pre** String matricolaDoc!=null && int tempo!=null && String tipo!=null **context** RicevimentoDAO:: public stampaRicevimenti(DocenteBean); **pre** String matricolaDoc!=null
Post- condizione	**context** RicevimentoDAO:: public stampaRicevimenti(DocenteBean) **post** stampaRicevimenti = true
Invarianti	

Nome classe	CorsoDAO
Descrizione	Questa classe rappresenta l'entità tabellare "Corso" contenuta all'interno del nostro DBMS relazionale. Ci permette di

	interfacciarci con il DBMS relazionale e di interrogarlo tramite dei metodi specifici.
Pre- condizione	**context** CorsoDAO:: public associaCorso(InsegnaBean); **pre** String codice!=null && String matricolaDoc=null **context** CorsoDAO:: public eliminaCorso(insegnaBean); **pre** String codice!=null && String matricolaDoc!=null **context** CorsoDAO:: public stampaCorsi(CorsoBean); **pre** String Codice!=null && String nome!=null && int num_CFU!=null && int ore_totali!=null **context** CorsoDAO:: public visualizzaCorsi(DocenteBean); **pre** String cognome!=null
Post- condizione	**context** CorsoDAO:: public stampaCorsi(CorsoBean); **post** stampaCorsi = true **context** CorsoDAO:: public visualizzaCorsi(DocenteBean); **post** visualizzaCorsi = true
Invarianti	

Nome classe	PrenotaDAO
Descrizione	Questa classe rappresenta l'entità tabellare "Prenota" contenuta all'interno del nostro DBMS relazionale. Ci permette di interfacciarci con il DBMS relazionale e di interrogarlo tramite dei metodi specifici.
Pre- condizione	**context** PrenotaDAO:: public effettuaPrenotazione(PrenotaBean); **pre** int numero_prenotazione!=null && String nome_corso!=null && Time orario!=null && Date giorno!=null && String tipologia!=null && String emailStud!=null && String matricolaDoc!=null

	context PrenotaDAO:: public eliminaPrenotazione (StudenteBean, int numprenot); **pre** int numero_prenotazione!=null && String emailStud!=null **context** PrenotaDAO:: public settaIndice(PrenotaBean); **pre** int i!=null **context** PrenotaDAO:: public settaTempo(PrenotaBean, ArrayList<RicevimentoBean>); **pre** String matricolaDoc!=null && Date giorno!=null && String tipo!=null **context** PrenotaDAO:: public getNumPrenotazioni(StudenteBean); **pre** String emailStud!=null **context** PrenotaDAO:: public getDocentePrenot(StudenteBean, int numprenot); **pre** String emailStud!=null && int numero_prenotazione!=null **context** PrenotaDAO:: public getDatiPrenot(StudenteBean, int numprenot); **pre** String emailStud!=null && int numero_prenotazione!=null **context** PrenotaDAO:: public getNumPrenotati(String matricoladocente) **pre** String matricolaDoc!=null **context** PrenotaDAO:: public listaPrenotatiDoc(PrenotaBean, DocenteBean) **pre** String matricolaDoc!=null
Post- condizione	**context** PrenotaDAO:: public getNumPrenotazioni(StudenteBean); **post** getNumPrenotazioni = true **context** PrenotaDAO:: public getDocentePrenot(StudenteBean, int numprenot); **post** getDocentePrenot = true **context** PrenotaDAO:: public getDatiPrenot(StudenteBean, int numprenot); **post** getDatiPrenot = true **context** PrenotaDAO:: public getNumPrenotati(String matricoladocente); **post** getNumPrenotati = true **context** PrenotaDAO:: public listaPrenotatiDoc(PrenotaBean, DocenteBean);

	post listaPrenotatiDoc = true
Invarianti	

Nome classe	PreferitiDAO
Descrizione	Questa classe rappresenta l'entità tabellare "Preferiti" contenuta all'interno del nostro DBMS relazionale. Ci permette di interfacciarci con il DBMS relazionale e di interrogarlo tramite dei metodi specifici.
Pre- condizione	**context** PreferitiDAO:: public getPreferiti (String preferitiEmailStud); **pre** String emailStud!=null **context** PreferitiDAO:: public deletePreferiti (String matricolaDoc, String preferitiEmailStud); **pre** String matricolaDoc!=null && String emailStud!=null **context** PreferitiDAO:: public setPreferiti (String preferitiEmailStud, String matricolaDoc); **pre** String emailStud!=null && String matricolaDoc!=null
Post- condizione	**context** PreferitiDAO:: public getPreferiti(String preferitiEmailStud); **post** getPreferiti= true
Invarianti	