**ODD**

[**1. Introduzione 2**](#_26ydc8ebauuc)

[1.1 Object design trade-offs 2](#_5imn2nkp4cwp)

[1.1.1 Robustezza vs Velocità di Implementazione 2](#_le3eied9ivri)

[1.2 Guidelines and conventions 2](#_pyb00clfyloc)

[1.3 References 2](#_prljd4wxn04e)

[2 Directory 2](#_j7rvsftzoqp6)

[2.1 src 2](#_5xycec9ethy5)

[2.2 db 2](#_fi5btsxr4lzi)

[2.3 WebContent 3](#_fia6ggawh2g3)

[2.3.1 icon 3](#_6vraueh8p620)

[2.3.2 META-INF 3](#_aozldq12uazg)

[2.3.3 style 3](#_j1qo231djaz)

[2.3.4 WEB-INF 3](#_3rb73ia8uet4)

[**3. Packages 3**](#_lh3jyb7cls34)

[3.1 view.Guest 3](#_d5r1x82iy855)

# Introduzione

## 1.1 Object design trade-offs

### 1.1.1 Robustezza vs Velocità di Implementazione

Nel contesto della gestione dei dati in ingresso, attribuiamo notevole importanza al controllo di tali dati. Tuttavia, effettuare questa operazione nel modo più accurato e completo possibile richiederebbe un investimento temporale superiore rispetto a quanto attualmente disponibile per il rilascio della prima versione, completamente funzionante. In considerazione di ciò, affermiamo una scelta che privilegia la tempestività di rilascio a discapito della robustezza iniziale. È importante notare che i controlli sui dati in ingresso saranno oggetto di attenzione e implementazione nelle versioni successive del sistema.

## 1.2 Guidelines and conventions

1. Le classi hanno dei nomi comuni singolari.
2. I metodi sono denominati con frasi verbali, i campi e i parametri con frasi nominali.
3. Gli Error Status sono restituiti attraverso eccezioni.

## 1.3 References

* RAD
* SDD

# 2 Directory

## 2.1 src

Questa directory deve contenere i pacchetti Java che compongono il codice del sistema.

## 2.2 db

Questa directory deve contenere lo schema del database relazionale.

## 2.3 WebContent

### 2.3.1 icon

Questa directory deve contenere le immagini utilizzate per icone o altro.

### 2.3.2 META-INF

Questa directory deve contenere meta informazioni.

### 2.3.3 style

Questa directory deve contenere i fogli di stile css.

### 2.3.4 WEB-INF

Questa directory contiene file essenziali per la Web Application come le librerie usate.

# 3. Packages

In questa sezione analizzeremo la suddivisione in pacchetti di tutte le classi implementate, ecco una prima panoramica del sistema implementato:

## 3.1 view.Guest

## 

# Design Pattern utilizzati

All’interno del nostro progetto sono stati utilizzati dei design pattern per riuscire a implementare alcune funzionalità del sistema.

Ad esempio abbiamo ritenuto utile andare a utilizzare un pattern di tipo interceptor per riuscire a realizzare il filtro che verifica i permessi per eseguire una determinata funzionalità. In questo modo andremo ad intercettare la richiesta e a scartarla nel caso in cui non si possiede il permesso necessario.

Il secondo pattern che abbiamo deciso di utilizzare è un pattern adapter/bridge per incapsulare la logica di pagamento. Quest’ultima infatti è realizzata attraverso dei servizi che non implementiamo direttamente noi, ma che ci vengono offerti da enti di terze parti. Risulta quindi cruciale per evitare di diventare dipendenti rispetto a quel servizio e la sua implementazione utilizzare un pattern per disaccoppiare.

All’interno del nostro sistema è necessario andare ad implementare dei dispatcher. Per individuare il validator al quale assegnare una nuova richiesta di pubblicazione, utilizzeremo proprio un dispatcher. Lo stesso discorso può essere fatto per il gestore catalogo e le richieste di rimozione.

A seconda di quella che è la mole di richieste di rimozione o di pubblicazione che sono presenti è necessario andare a mutare quella che è la strategia di distribuzione delle richieste. Sembra esattamente il problema risolto dallo Strategy.