

Design Pattern Document

| **Riferimento** | C07\_DPD\_ver1.0 |
| --- | --- |
| **Versione** | 1.0 |
| **Data** | 08/01/2025 |
| **Destinatario** | Prof.ssa F. Ferrucci, Prof. F. Palomba |
| **Presentato da** | C07 |
| **Approvato da** |  |

# 

# Revision History

| Data | Versione | Descrizione | Autore |
| --- | --- | --- | --- |
| 05/01/2025 | 0.1 | Stesura Iniziale documento | EF |
| 08/01/2025 | 0.2 | Revisione finale documento | EF, ELA, MLR, PL, AP, AC, GS, GR |
| 16/01/2025 | 0.3 | Revisione formattazione | MLR |

# 

# Members

| Nome | Ruolo nel progetto | Acronimo | Informazioni di contatto |
| --- | --- | --- | --- |
| Alessandro Zoccola | Project Manager | AZ | a.zoccola2@studenti.unisa.it |
| Paolo Carmine Valletta | Project Manager | PV | p.valletta2@studenti.unisa.it |
| Alessandro Pinto | Team Member | AP | a.pinto44@studenti.unisa.it |
| Andrea Carbone | Team Member | AC | a.carbone89@studenti.unisa.it |
| Emanuele Falanga | Team Member | EF | e.falanga3@studenti.unisa.it |
| Emanuele Luigi Amore | Team Member | EA | e.amore2@studenti.unisa.it |
| Gabriele Ristallo | Team Member | GR | g.ristallo3@studenti.unisa.it |
| Giuseppe Sica | Team Member | GS | g.sica53@studenti.unisa.it |
| Maurilio La Rocca | Team Member | MR | m.larocca31@studenti.unisa.it |
| Pasquale Livrieri | Team Member | PL | p.livrieri@studenti.unisa.it |

# **Sommario**

[**Revision History 2**](#_8b1bhwiax10r)

[**Team Members 3**](#_8nyy1bgp8f4v)

[**Sommario 4**](#_8ke5pkrsof7t)

[**1 - Introduzione 5**](#_sscew2ocxvd1)

[**2 - Scopo del documento 5**](#_uaw7r6fthf4j)

[**3 - Relazione con altri documenti 5**](#_m87ylxmb89hm)

[**4 - Design Pattern 6**](#_cp0toybwcnzn)

[4.1 - Adapter 6](#_hmkjqi1obu4e)

[4.2 - Factory Method 7](#_etz022z000eq)

[4.3 - Facade 8](#_nb8qfg74wfmb)

# **1 -** Introduzione

L’obiettivo di questo documento è presentare possibili punti dove applicare design pattern all’interno del progetto “MindArt”.

# **2 -** Scopo del documento

Questo documento si pone di proporre tre design pattern. Per ognuno di questi saranno descritti:

* Locazione della modifica
* Razionale della modifica;
* Schema UML del design pattern applicato;

# **3 -** Relazione con altri documenti

Questo documento è strettamente correlato con altri documenti di progetto, tra cui:

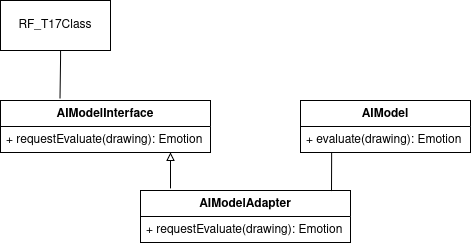
* [Matrice di Tracciabilità](https://docs.google.com/spreadsheets/u/1/d/1QXW_d-vO6Kufg2dqaf9zHAGH_WsH6681XLRMKv2Rp-M/edit);
* [System Design Document](https://docs.google.com/document/u/1/d/1J2AmgPDhh2XV7CGk3ndgWdhtSZIaMktT1ty_o-KNt9w/edit).

# 

# **4 -** Design Pattern

### 4.1 - Adapter

* **Locazione**: RF\_T17 - funzionalità non implementata;
* **Razionale**: il requisito RF\_T17 prevede un sistema di intelligenza artificiale a supporto del terapeuta per l’identificazione dei sentimenti del bambino sulla base dei colori utilizzati.  
  E’ plausibile che il suddetto requisito venga realizzato tramite un modulo esterno, per esempio in Python.  
  Il Design Pattern “Adapter” viene in aiuto per l’integrazione di questo modulo fornendo al sistema corrente un’interfaccia che astragga i compiti di richiesta verso il modulo esterno.
* **Diagramma**:



### 

### 4.2 - Factory Method

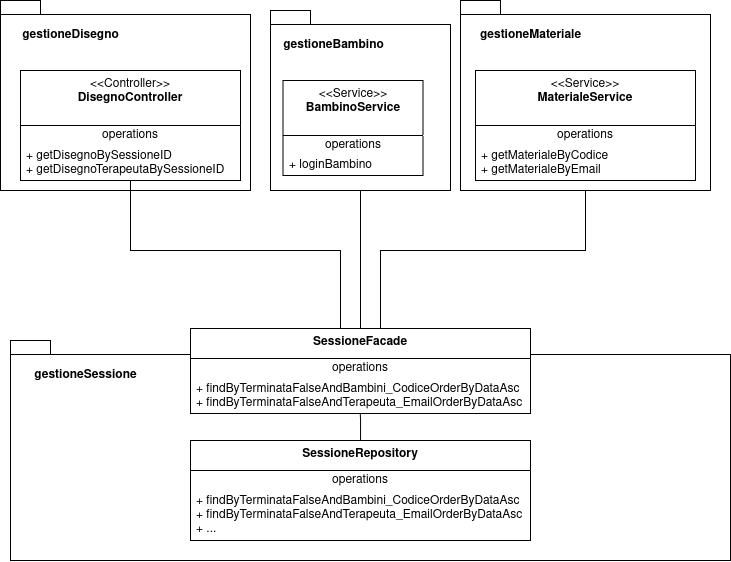
* **Locazione**: classe *com.is.mindart.gestioneMateriale.controller.MaterialeController* relativa al requisito RF\_T11.
* **Razionale**: al momento la logica di creazione di un oggetto Materiale (in particolare nel metodo *addMaterial*) è fatta tramite codice poco leggibile e manutenibile: qualsiasi modifica ai tipi di file supportati o alla logica di acquisizione di questi causerebbe una serie di cambiamenti sia nelle classi relative al model del materiale che a questa. Il pattern “Factory Method” è un pattern creazionale che permette di spostare la logica di creazione di diversi “Prodotti” in una particolare classe “Factory”. Verrebbe quindi utilizzata questa factory nel controller in questione per creare l’oggetto materiale.  
  I punti a favore sarebbero:
  + Single Responsibility Principle: la responsabilità della creazione del materiale viene assegnata ad una componente precisa, aumentando la manutenibilità del sistema;
  + Open/Closed Principle: si possono aggiungere, rimuovere o modificare i materiali supportati senza richiedere modifiche dalle componenti che utilizzano la Factory;
  + Riduce l’accoppiamento tra il controller e il materiale.
* **Diagramma**:

### 

### 

### 4.3 - Facade

* **Locazione**: qualsiasi package
* **Razionale**: Allo stato attuale è comune che una classe di un certo package utilizzi una repository o una service locata in un altro package. Questo causa eccessivo accoppiamento tra classi di package separati. Per ridurlo è possibile utilizzare il pattern “Facade”. Questo permette di esporre un’interfaccia contenente i servizi esposti dal package, non rendendo più necessario riferirsi direttamente alle classi del package in questione.  
  Ha quindi i vantaggi di:
  + Ridurre di molto l’accoppiamento;
  + Aumentare la manutenibilità complessiva del codice;
  + Nascondere la complessità di un package agli utilizzatori;
* **Diagramma**: Esempio relativo a *com.is.mindart.gestioneSessione*

****