Object Design Document

Progetto:

CleanDesk

CleanDesk
Powered by AI

Riferimento:	
Versione:	0.6
Data:	10/02/2024
Destinatario:	Esame di Ingegneria del Software 2023/24
Presentato da:	Ambrosio Gennaro, Camoia Andrea
Approvato da:	

UNIVERSITÀ DEGLI STUDI
DI SALERNO

Revision Hystory

Data	Versione	Descrizione	Autori
27/12/2023	0.1	Stesura iniziale del documento	Ambrosio Gennaro Camoia Andrea
08/01/2024	0.2	Specifica dei Package	Ambrosio Gennaro Camoia Andrea
10/01/2024	0.3	Specifica Interfacce delle Classi	Ambrosio Gennaro Camoia Andrea
13/01/2024	0.4	Specifica del Design Pattern	Ambrosio Gennaro Camoia Andrea
08/02/2024	0.5	Aggiunta Glossario	Ambrosio Gennaro Camoia Andrea
10/02/2024	0.6	Aggiustamenti Generali	Ambrosio Gennaro Camoia Andrea

Team Member

Nome	Ruolo nel progetto	Acronimo	Informazioni di contatto
Andrea Camoia	Team Member	AC	a.camoia@studenti.unisa.it
Gennaro Ambrosio	Team Member	AG	g.ambrosio35@studenti.unisa.it

Indice

1 Introduzione

CleanDesk è un software intelligente progettato per **ottimizzare** l'esperienza informatica e **aumentare la produttività** di una vasta gamma di utenti. La nostra App organizza e facilita la ricerca di documenti, immagini e progetti all'interno di qualsiasi cartella e ambiente digitale, come il Desktop. oInoltre, CleanDesk offre un'**efficace gestione e analisi della memoria**, consentendo agli utenti di utilizzare consapevolmente il proprio spazio di archiviazione.

Questa sezione del documento fornirà istruzioni sulla scrittura del codice, elencherà definizioni, acronimi e abbreviazioni utilizzati nell'intero testo ed infine presenterà riferimenti e link utili.

1.1 Linee Guida per la Documentazione delle Interfacce

In questo paragrafo saranno illustrate le regole che dovranno essere rispettate durante l'implementazione del sistema. Di seguito verrà riportata una lista delle convenzioni utilizzate nella stesura del codice, con link ufficiali inclusi.

1.1.1 Documentazione ufficiale

- **Electron:** `https://www.electronjs.org/docs/latest`
- **HTML:** `https://www.w3schools.com/html/html_intro.asp`
- **Javascript:** `https://devdocs.io/javascript/`
- **Python:** `https://docs.python.org/3/`

1.1.2 Interfacce in JavaScript

Come già specificato, l'implementazione di CleanDesk sarà fatta con **JavaScript**. Tuttavia, quest'ultimo è un **linguaggio di programmazione non tipizzato**, il che rappresenta una sfida significativa nell'implementazione e nella specifica delle interfacce delle classi.

Le interfacce delle classi definiscono il contratto che una classe deve seguire, includendo i metodi e le proprietà che devono essere implementati. Tuttavia, a differenza di linguaggi con tipi statici come Java, JavaScript non offre un sistema di tipi statico incorporato per garantire che una classe implementi correttamente un'interfaccia.

Di conseguenza, la specifica delle interfacce descritta in questo documento, non potrà essere riportata nell'implementazione e va presa come **riferimento e linea guida** per la futura stesura del codice.

1.2 Definizioni, Acronimi e abbreviazioni

Di seguito sono definite e specificate le definizioni, acronimi e abbreviazioni utilizzate all'interno di questo documento:

- **Design pattern:** template di soluzioni a problemi ricorrenti impiegati per ottenere riuso e flessibilità.
- **Packages:** Insieme di file, interfacce e classi correlate e relative ad un software.
- **Interfaccia:** definisce i metodi, le operazioni e le regole d'utilizzo di una classe;
- **Interface Layer:** nel pattern Three-layer include tutti i boundary object che interfacciano con l'utente.
- **Application Logic Layer:** nel pattern Three-layer, include tutti gli oggetti relativi al controllo e alle entità che realizzazione l'elaborazione, le regole di verifica e notifica richiesta dall'applicazione.
- **Storage Layer:** nel pattern Three-layer, effettua la memorizzazione, il recupero e l'interrogazione di oggetti per-sistenti.

1.3 Riferimenti

Di seguito vengono riportate le risorse utilizzate per la stesura della presente documentazione ed utili inoltre alla sua lettura:

- CleanDesk Statement of Work (SOW)
- CleanDesk Requirements Analysis Document (RAD)
- CleanDesk System Design Document (SDD)
- CleanDesk Test Plan (TP)
- CleanDesk Test Case Specification (TCS)
- Dispense del prof. Carmine Gravino, fornite mediante la pagina del corso "Ingegneria del Software - Resto 2 - 2023/2024" sulla Piattaforma E-learning del Corso di Laurea in Informatica dell'Università degli Studi di Salerno;
- Libro di testo "Object Oriented Software Engineering Using UML Patterns and Java Prentice Hall 2010 Bernd Bruegge Allen H.Dutoit"
- Libro di testo "C. GHEZZI, D. MANDRIOLI, M. JAZAYERI, INGEGNERIA DEL SOFTWARE – FONDAMENTI E PRINCIPI, PRENTICE HALL, 2004"

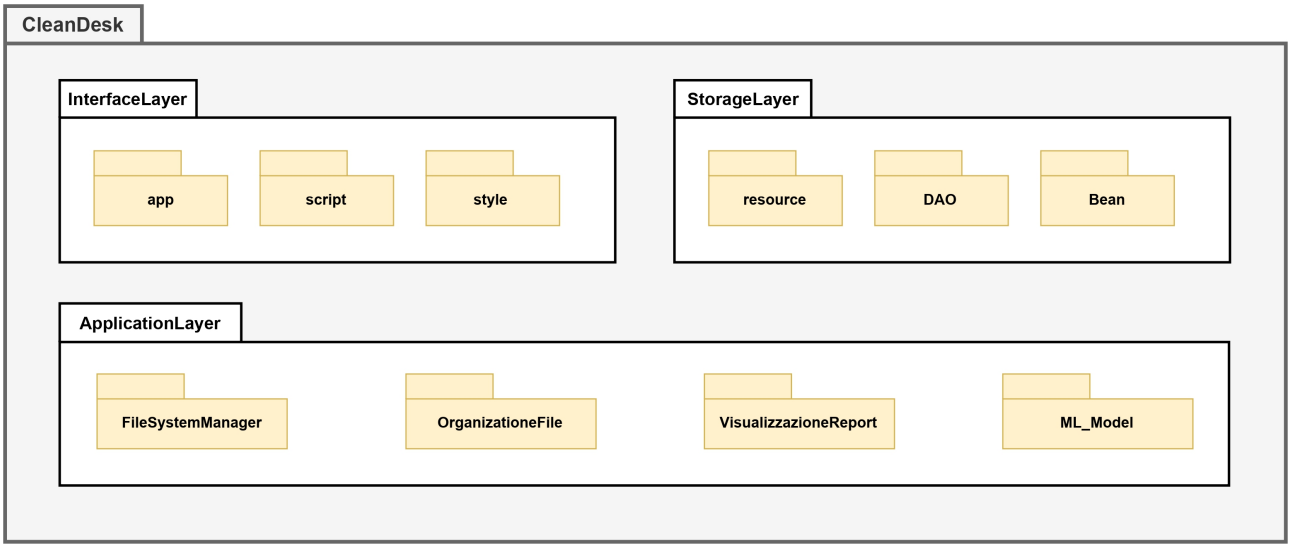
2 Packages

In questa sezione viene mostrata la suddivisione del sistema in package, in base a quanto definito nel documento di System Design.

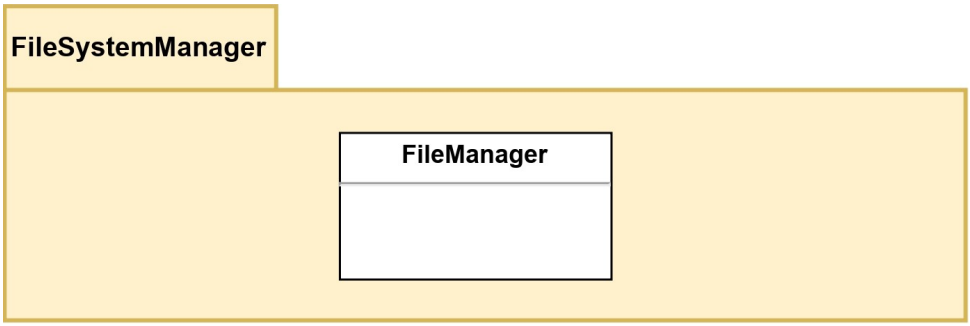
- `node_modules`: contiene tutti i moduli e le dipendente di **node.js** ed **electron**.
- `src`: contiene i **file sorgente**.
  - `InterfaceLayer`: contiene tutti gli **oggetti boundary** e i file utili per l'interfaccia utente
  - `StorageLayer`: contiene file in codice `JavaScript` per la gestione degli **oggetti model** e la connessione con il database
    - → `resource`: contiene il file del database utilizzato per il salvataggio degli oggetti model
  - `ApplicationLayer`: contiene tutti i file per l'implementazione degli **oggetti control** del programma, e un file in codice `Python` per l'accensione del server.
    - → `FileSystemManager`: contiene file in codice `JavaScript` per la gestione del File System
    - → `ML_Model`: contiene file in codice `Python` relativi al modello di ML
    - → `OrganizzazioneFile`: contiene file per la logica del control per la funzionalità di Classificazione
    - → `VisualizzazioneReport`: contiene file in codice `JavaScript` utili per la logica della funzionalità di visualizzazione dei Report precedenti
- `test`: contiene tutti i file necessari al **testing**.

2.1 Package di CleanDesk

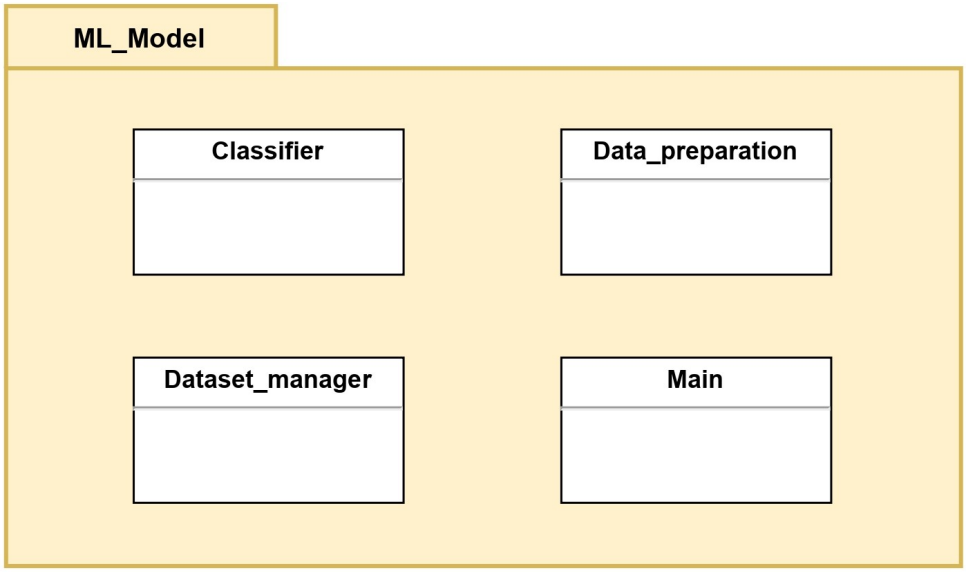
Mostriamo ora la **suddivisione dell'intero sistema in package**. Questa divisione è stata fatta sulla base dell'architettura software del sistema e delle scelte fatte durante la fase di System Design.



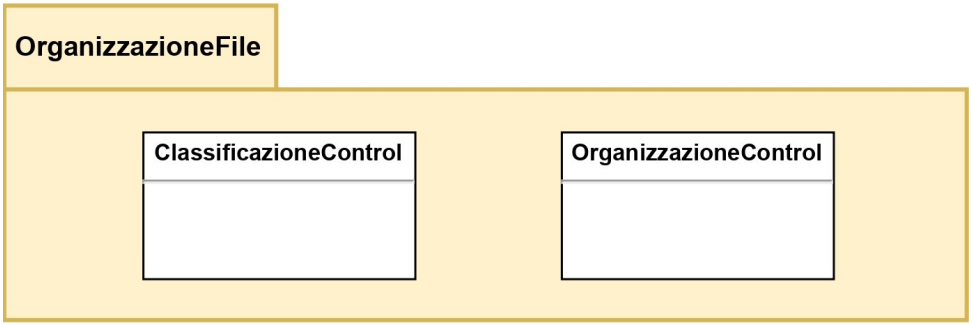
2.1.1 Package FileSystemManger



2.1.2 Package ML_Model



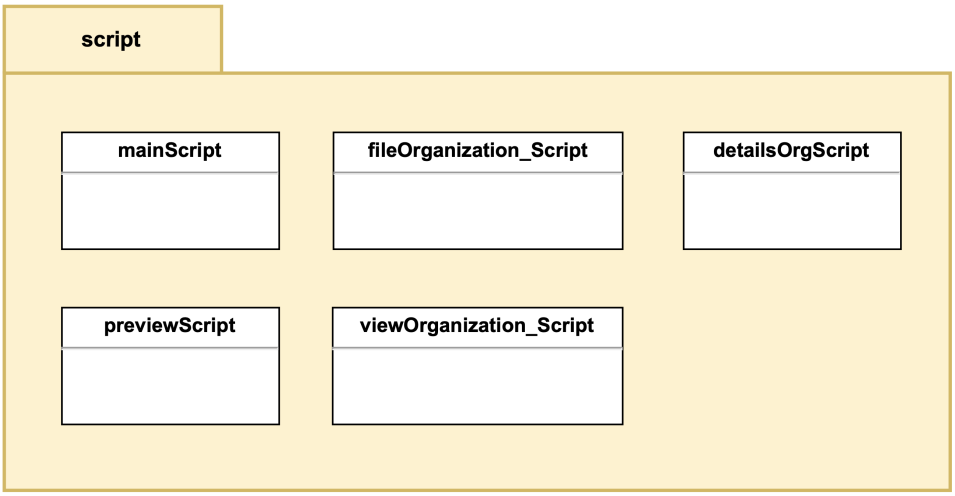
2.1.3 Package OrganizzazioneFile



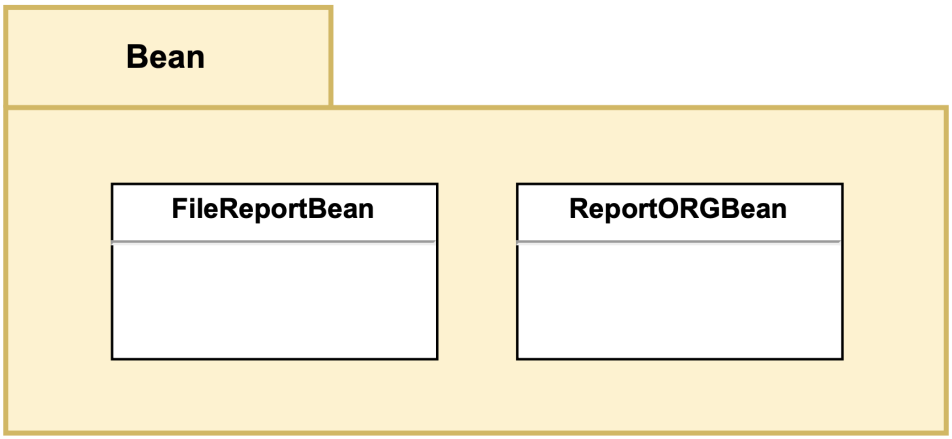
2.1.4 Package VisualizzaReport

2.1.5 Package app

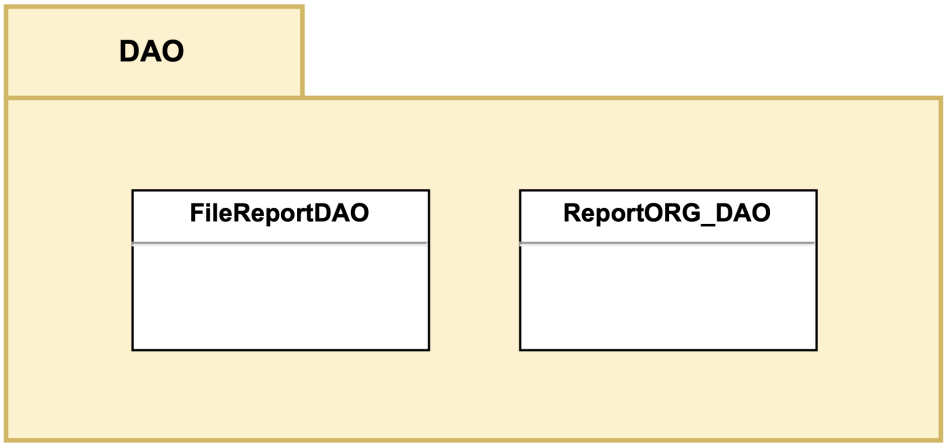
2.1.6 Package script



2.1.7 Package bean



2.1.8 Package DAO



3 Class Interfaces

3.1 Specifica Interfacce del Sistema

In questa sezione del documento saranno mostrate alcune delle interfacce del sistema. In particolare sarà fornita un Interfaccia per ogni package definito e a sua volta un metodo per ognuna di questa.

3.1.1 FileSystemManager Interfaces

Nome Classe	FileManager
Descrizione	Questa classe gestisce le funzionalità di accesso, creazione, rimozione e modifica di File e Cartelle all'interno del File System.
Metodi	+ async createDir(*List<FileReport>* listFileReoport) + async moveFile(*int* reportORG_ID)
Invariante	//

Nome Metodo	+ async moveFile(*int* reportORG_ID)
Descrizione	Questo metodo permette di spostare la posizione di tutti i file riportati nel report organizzazione.
Pre-Condizione	reportORG_ID != null and reportORG_ID.isInDatabase() == true
Post-Condizione	//

3.1.2 OrganizzazioneFile Interfaces

Nome Classe	OrganizzazioneControl
Descrizione	Questa classe gestisce le funzionalità di creazione, modifica, rimozione di un'organizzazione file avviata dall'utente.
Metodi	+ async creaReportORG(*JSON Log* logEntries, *String* initPath) + async deleteReportORG(*int* reportID) + async updateReportInfo(*int* reportID, *int* name, *string* description)
Invariante	//

Nome Metodo	+ async creaReportORG(*JSON Log* logEntries, *String* initPath)
Descrizione	Questo metodo crea l'organizzazione vera e propria, salvando tutte le informazioni necessarie ed i singoli FileReport.
Pre-Condizione	Array.isArray(logEntries) == true logEntries.length != 0 logEntries.length != null

Specifica JSON	Di seguito specifichiamo la struttura di **JSON Log**, ovvero la struttura dati in cui sarà contenuta la classficazione dei file fornita dal modello di Machine Learning:
	
	`[`
	`  ...`
	`  {`
	`  "fileName" : "file",`
	`  "filePath" : "/usr/oldfolder/file.txt",`
	`  "finalPath": "/usr/oldfolder/file.txt",`
	`  "category" : "Sport"`
	`  }`
	`  ...`
	`]`

Post-Condizione	//

3.1.3 VisualizzaReport Interfaces

Nome Classe	VisualizzaReportControl
Descrizione	Questa classe gestisce le funzionalità di visualizzazione delle organizzazioni effettuate.
Metodi	+ async getAllReports(): List<ReportORG> + async getUntilReport(*Date* dateTo): List<ReportORG> + async getFromToReport(*Date* dateFrom,*Date* dateTo):List<ReportORG> + async viewDetailsByReportID(*int* reportID): List<FileReport> + async getReportByID(*int* reportID): ReportOrg + async getAllFromReports(*Date* dateFrom): List<ReportORG>
Invariante	//

Nome Metodo	+ async getAllReports(): List<ReportORG>
Descrizione	Questo metodo permette di prelevare tutti i Report Organizzazione presenti nel database.
Pre-Condizione	//
Post-Condizione	//

Nome Metodo	+ async viewDetailsByReportID(*int* reportID): List<FileReport>
Descrizione	Questo metodo permette di visualizzare tutti i FileReport presenti in un ReportOrganizzazione.
Pre-Condizione	reportID.isInDatabase() == true
Post-Condizione	//

3.1.4 Bean Interfaces

Nome Classe	ReportORGBean
Descrizione	Questa classe rappresenta l'entità Report Organizzazione.
Metodi	+ setters + getters
Invariante	//

Nome Metodo	+ getters
Descrizione	Insieme di metodi che restituiscono gli attributi del Report Organizzazione.
Pre-Condizione	//
Post-Condizione	//

Nome Metodo	+ setters
Descrizione	Insieme di metodi che permettono di modificare il valore degli attributi del Report Organizzazione.
Pre-Condizione	//
Post-Condizione	ReportORGBean.[attributo] = [newValue]

3.1.5 DAO Interfaces

Nome Classe	FileReportDAO
Descrizione	Questa fornisce l'accesso al database per gestire le informazioni dei FileReport.
Metodi	+ async getAll(): List<FileReport> + async getAllByReportID(*int* reportID): List<FileReport> + async saveFileReport(*FileReport* fileReport): *int* lastID + async removeAll()
Invariante	databaseConnection != null

Nome Metodo	+ async saveFileReport(*FileReport* fileReport): *int* lastID
Descrizione	Metodo che permette di salvare un nuovo FileReport nel database.
Pre-Condizione	fileReport != null
Post-Condizione	//

4 Class Diagram Ristrutturato

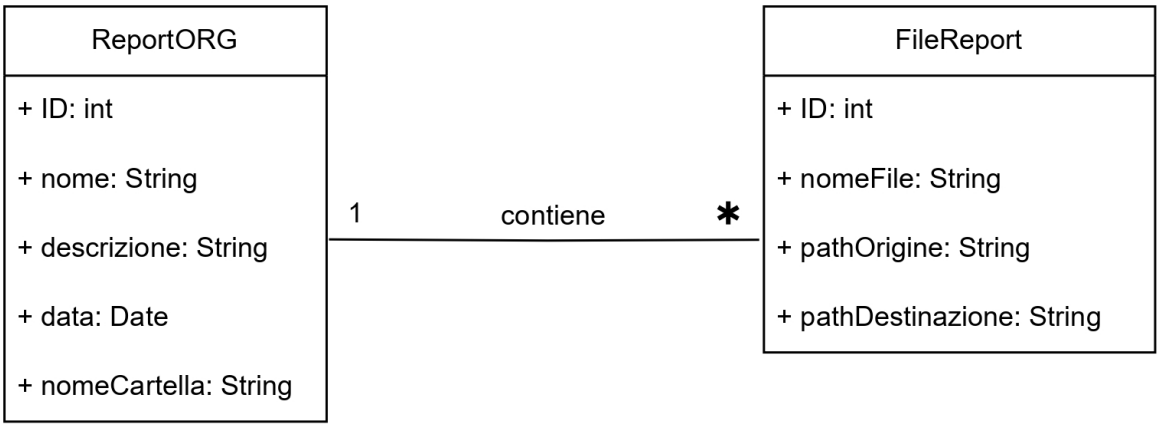


Figure 4.1: Ristrutturazione del Class Diagram

5 Elementi di Riuso

In quest'ultima sezione sono specifica le **tecniche di Riuso** utilizzate per l'implementazione.

5.1 Design Pattern

In particolare, all'interno del sistema è stato utilizzato un Design Pattern per risolvere un problema di implementazione e per semplificare il funzionamento dell'applicazione.

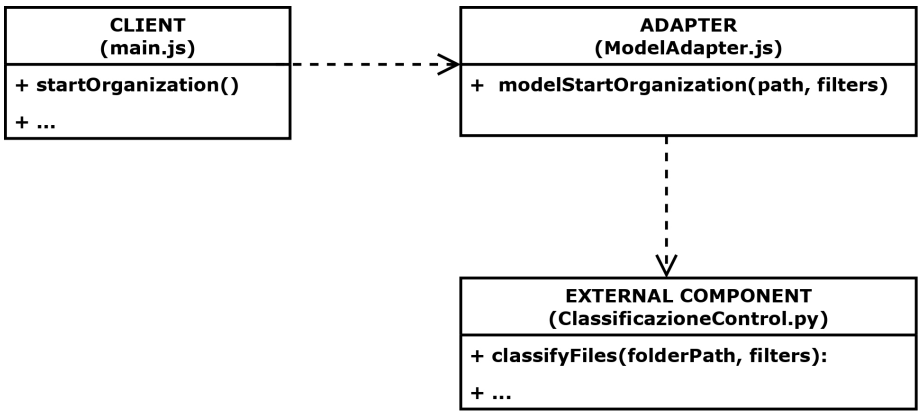
5.1.1 Adapter

Il fulcro di CleanDesk è rappresentato dalla **componente di Machine Learning**, che permette di effettuare la classificazione vera e propria dei file. Tuttavia, il classificatore è implementato in Python e di conseguenza è impossibile farlo comunicare direttamente con il sistema principale realizzato in `Electron JS`.

Per risolvere questo problema è stato utilizzato **l'Adapter design pattern**, ovvero un patter strutturale che consente ad oggetti con interfacce incompatibili di collaborare. L'oggetto adapter quindi andrà ad interporsi tra il client ed il sistema da integrare, in modo tale da convertire le richieste che vengono fatte a quest'ultimo e allo stesso tempo restituendo i suoi servizi in maniera comprensibile al sistema cliente.

Questo design pattern, nel nostro sistema, è stato realizzato mediante la creazione di una **nuova classe** `"ModelAdapter"`, alla quale si interfaccerà il software CleanDesk per avviare una nuova organizzazione. La **richiesta al servizio esterno**, ovvero il modulo di Machine Learning, avviene tramite il metodo `"modelStartOrganization(path, filters)"` della classe adapter, che tramite una **fetch** ad un server locale (implementato sempre con python) permette di **avviare la classificazione** e ricevere il log risultante.

Mostriamo di seguito una lo schema del funzionamento applicato:



6 Glossario

Sigla/Nome	Descrizione
Package	Insieme di Classi o File di Codice Sorgente.
Machine Learning	Una branca dell'intelligenza artificiale che si occupa dello sviluppo di algoritmi e modelli che consentono ai computer di imparare dai dati forniti, senza essere esplicitamente programmati per compiti specifici.
DAO	Un Data Access Object, è un design pattern architetturale utilizzato per separare la logica di accesso ai dati dalla logica di business all'interno di un'applicazione.
Bean	Semplice classe che rappresenta un entità del problema.
Design Pattern	Sono template di soluzioni a problemi ricorrenti impiegati per ottenere riuso e flessibilità.
Tree-Layer	Modello di organizzazione del codice di un software basato sulla stratificazione dei sottosistemi in 3 livelli indipendenti.