Documento Di Design

1 Analisi Architettura

Il software sviluppato si presenta come un'architettura client-server.Un sottosistema ,detto server, fornisce servizi ad istanze di altri sottosistemi detti client che sono responsabili dell'interazione con l'utente e mettono a disposizione un'interfaccia grafica semplice e intuitiva. Questa soluzione è comunamente addotata per le applicazioni che richiedono una gestione centralizzata dei dati garantendo che tutte le informazioni e i dati persistenti siano conservati e aggiornati sul server

1.1 Client

Il client è stato sviluppato utilizzando JavaFX, una scelta motivata principalmente dal desiderio di creare un'applicazione desktop con un'interfaccia grafica accattivante. Allo stesso tempo, volevamo apprendere una nuova tecnologia, distaccandoci dal progetto di Object Orientation. Abbiamo optato per JavaFX sviluppando il front-end e utilizzando strumenti come SceneBuilder per semplificare l'inserimento e l'aggiornamento dei componenti all'interno dell'interfaccia grafica.

E' stato scelto il pattern MVC(Model-View-Controller), la scelta ricade in questo pattern architteturale perchè presenta diversi vantaggi:

- **Separazione Delle Responsabilità:**MVC permette una chiara separazione delle responsabilità fra presentazione grafica ed interattiva(View), gestione del modello dei dati di interesse(Model) e una definizione della logica di controllo e delle funzionalità applicative(Controller).
- **Estendibilità:**MVC permette una semplicità di progetto e decentralizzazione dell'architettura avendo un software facilmente estendibile agendo su moduli specifici.
- **Manutenibilità:**MVC permette la possibilità di intervenire su componenti specifici con effetti nulli o limitati su altri componenti.

Abbiamo deciso di adottare in concomitanza con il Design Pattern Architteturale MVC,il Design Pattern Repository per comunicare con il back-end e promuovere il Principio della Singola Responsabilità. Il Repository è un design pattern che si occupa della gestione dell'accesso ai dati, effettuando richieste HTTP tramite REST API. Esso funge da intermediario tra il livello di business e il back-end dell'applicazione, fornendo un'interfaccia uniforme per l'accesso e la manipolazione dei dati, garantendo così una chiara separazione delle responsabilità.

Ogni Controller è progettato con una dipendenza verso uno strato aggiuntivo, denominato **Repository Layer**. Questo layer è responsabile della comunicazione con il Back-End tramite **API REST**, inviando e ricevendo richieste HTTP.

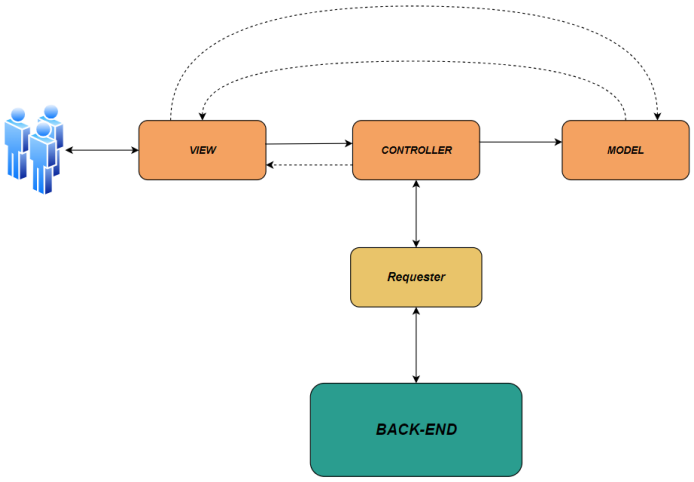
L'introduzione del Repository Layer consente di separare in modo netto la logica dell'applicativo presente nei Controller dalla gestione delle comunicazioni con il back-end. In questo modo, ogni modulo dell'applicazione mantiene una chiara responsabilità, favorendo una maggiore manutenibilità, scalabilità e riusabilità del codice.

Inoltre, questa architettura rende più semplice il testing del Controller e del Repository in modo indipendente, garantendo che le modifiche al livello di comunicazione non impattino direttamente sulla logica di business gestita dai Controller. Questo approccio aderisce ai principi di **Single Responsibility** e **Separation of Concerns**, fondamentali per la progettazione di applicazioni robuste e modulari.

1.1.1 Architettura Front-End

Si è scelto di suddivivere l'architettura del Front-End in 4 layer: Model,View,Controller,Repository.

- **Model:**Questo package si occupa di contenere il **Modello Dei Dati** di interesse all'applicazione e ha il compito di notificare alla view i cambiamenti di stato.
- **View:**Questo package fornisce una rappresentazione grafica ed interrativa del model. Definisce le modalità di rappresentazione dei dati, consente l'interazione con l'utente e riceve notifiche dal model, in modo da aggiornare la sua visualizzazione.Può interrogare il Model per chiedere informazioni riguardanti il suo stato
- **Controller:**Questo package definisce la logica di controllo e le funzionalità applicative,gestisce gli eventi ed i comandi generati dall'utente, opera sul model in base ai comandi ricevuti,può selezionare la view adatta in base ai comandi fatti dall'utente e può chiamare il layer Requester nel caso l'applicativo debba comunicare con il back-end.
- **Repository:**Questo package si occupa di comunicare con il back-end facendo delle richieste HTTP,ricevendo delle HTTP response e comunicarle al controller.



1.1.2 Front-End Struttura Schematizzata

Con questo schema viene riassunto il modo di comunicare fra i vari componenti del front-end, rendendo la visione del suo funzionamento più semplice e chiara

1.2 Server

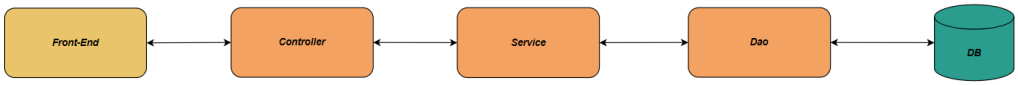
Il server è stato sviluppato usando Jakarta RESTful Web Services, è un framework Java che fornisce un'API per la creazione di servizi Web RESTful. REST (Representational State Transfer) è uno stile architetturale per la progettazione di servizi Web che si basa sui principi fondamentali della trasmissione dei dati tramite HTTP.

1.2.1 Architettura Back-End

Si è scelto di suddividere l'architettura Back-End in 3 layer: Controller,Service,DAO.

- **Controller:** Questo package si occupa di fare il routing delle richieste HTTP e di gestire le richieste ricevute e da inviare al Client.
- **Service:** Questo package si occupa della logica di business dell'applicazione,offre una serie di funzionalità che verranno richiamate dal controller per eseguire azioni all'interno del Back-End.
- **DAO:** Questo package è responsabile dell'accesso alla base di dati. Contiene metodi **CRUD**, che permettono di inserire,leggere,aggiornare e cancellare dati nella base di dati. La sua implementazione è stata fatta attraverso JDBC.

1.2.2 Struttura Schematizzata Back-End



Con questo schema viene riassunto il modo di comunicare fra i vari componenti del Back-End, rendendo la visione del suo funzionamento più semplice e chiara