JBudget

Francesco Palozzi

2025

Contents

1 Introduzione Progetto

Nella progettazione, ho dato particolare rilievo ai concetti chiave dell'OOP come l'incapsulamento, l'ereditarietà e il polimorfismo, cercando di creare un'architettura che fosse al tempo stesso solida ed estensibile. L'approccio MVC (Model-View-Controller) è stato l'ideale proprio perché permette di separare chiaramente le responsabilità, facilitando sia lo sviluppo iniziale che eventuali modifiche future.

La scelta delle tecnologie - Java 21 come linguaggio principale, Gradle per la gestione delle dipendenze, JavaFX per l'interfaccia grafica e Hibernate con PostgreSQL per la persistenza dei dati - è stata guidata dalla necessità di creare un'applicazione robusta che potesse servire come base per ulteriori sviluppi. Ogni componente è stato pensato non solo per soddisfare i requisiti attuali, ma anche per agevolare l'integrazione di nuove funzionalità in un secondo momento.

1.1 Funzionalità Principali

L'applicazione è stata progettata per fornire un insieme completo di strumenti per la gestione del budget familiare, con particolare attenzione alla flessibilità e all'usabilità. Le funzionalità implementate costituiscono una solida base che può essere estesa in futuro per arricchire l'esperienza utente.

- Gestione delle transazioni finanziarie

- Pianificazione finanziaria
- Analisi e Reporting

Queste funzionalità sono state sviluppate con particolare attenzione all'usabilità, garantendo un'interfaccia intuitiva che rende semplice la gestione quotidiana delle finanze domestiche, pur offrendo strumenti avanzati per chi desidera un controllo più approfondito del proprio budget.

1.2 Suddivisione Responsabilità

L'architettura del sistema è stata progettata seguendo rigorosamente il pattern Model-View-Controller (MVC), una scelta che garantisce una chiara separazione delle competenze tra i diversi componenti dell'applicazione. Questa strutturazione permette non solo una migliore organizzazione del codice, ma facilita anche la manutenzione e l'estensione futura del progetto.

- Adattabilità per diversi dispositivi
- Sincronizzazione tra diversi dispositivi
- Persistenza delle informazioni e dei dati
- Gestione interazioni utente-interfaccia
- Testabilità delle diversi componenti

2 Funzionalità Implementate

L'applicazione offre un insieme completo di strumenti per la gestione finanziaria personale e familiare. Al centro del sistema troviamo una solida gestione delle transazioni finanziarie, che permette di registrare ogni movimento con la possibilità di categorizzarlo e assegnargli tag personalizzati per una classificazione dettagliata. Particolarmente utile risulta la gestione delle transazioni ricorrenti, che automatizza la registrazione di spese e entrate periodiche come affitti, stipendi o abbonamenti.

Il modulo di scadenze finanziarie aiuta gli utenti a tenere sotto controllo i pagamenti futuri, visualizzandoli in una pratica lista. Per quanto riguarda l'analisi dei dati, l'applicazione genera statistiche visuali chiare ed immediate, con grafici che mostrano l'andamento delle entrate e uscite nel tempo. Un sistema di filtraggio avanzato permette poi di selezionare e analizzare specifiche tags di transazioni o periodi temporali, offrendo così una visione personalizzata della situazione finanziaria.

Tutte queste funzionalità lavorano in sinergia per fornire un quadro completo e

aggiornato della salute economica familiare, trasformando dati grezzi in informazioni utili per prendere decisioni finanziarie consapevoli. L'interfaccia intuitiva rende semplice l'utilizzo quotidiano, pur mantenendo la potenza necessaria per analisi più approfondite.

2.1 Gestione Gruppo

Il programma è composto da diversi *Gruppi*. Ogni gruppo può contenere diversi *Account* e *Categorie* (utilizzate per categorizzare le transazioni). Ed ogni categoria può contenere diversi *Tag* (utilizzati per etichettare le transazioni)

Per la creazione o eliminazione di un gruppo bisogna trovarsi nel tab *Dashboard*

Un singolo gruppo ha bisogno di un *Nome* e di una *Currency*. La currency selezionata sarà utilizzata per tutte le transazioni di quel gruppo e per il bilancio iniziale di ogni account associato a quel gruppo.

L'eliminazione del gruppo causerà automaticamente l'eliminazione di tutti gli *Account*, *Transazioni*, *Categorie* e *Tag* associati ad esso

2.2 Gestione Transazioni Finanziarie

Dopo aver selezionato un gruppo è possibile creare, eliminare e aggiornare transazioni attraverso la tab *Transaction*.

2.2.1 Creazione Transazione

Per la creazione di una nuova transazione bisogna cliccare il bottone *New Transaction*, alla quale seguirà l'apertura di un dialog. In questo dialog si sceglie se si tratta di una transazione ricorrente o meno.
Quindi una volta cliccato il bottone no si aprirà il dialog per la creazione effettiva della transazione.

Qui bisognerà aggiungere le informazioni essenziali di una transazione e, se c'è né il bisogno, sarà possibile creare nuovi account, categorie e tags. Una volta creata la nuova transazione questa apparirà nella tabella delle transazioni (Se la data inserita si trova all'interno delle date nel filtro)

2.2.2 Eliminazione Transazione

Per eseguire l'eliminazione di una transazione bisognerà prima di tutto selezionare la transazione da eliminare dalla tabella, e successivamente cliccare il bottone *Delete Transaction*

2.2.3 Aggiornamento Transazione

Per aggiornare una transazione bisognerà prima di tutto selezionare la transazione da aggiornare dalla tabella, e successivamente cliccare il bottone *Update Transaction*. Alla quale seguirà l'apertura di un dialog con i dati della transazione da modificare

2.3 Gestioni Transazioni Ricorrenti

La creazione di una transazione ricorrente è la medesima di una transazione normale. L'unica differenza è che al posto della data dobbiamo scegliere la data di inizio e di fine, ed in più la ricorrenza della transazione.

Questa funzionalità risulta comoda per rendere più veloce e semplice l'aggiunta di transazioni ricorrenti come per esempio uno stipendio o un abbonamento.

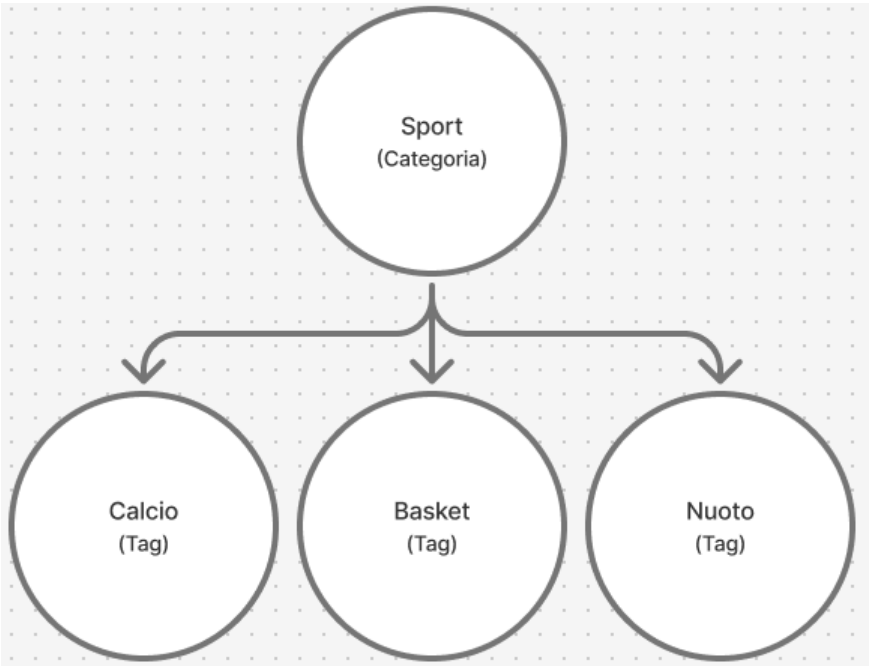
2.4 Gestione Scadenze Finanziarie

Dopo aver selezionato un gruppo è possibile creare, eliminare e aggiornare lo scadenziario attraverso la tab *Scheduler*. (Proprio come per le transazioni normali - Non sono presenti scadenze ricorrenti)

2.5 Tagging e Categorizzazione Transazioni

Il sistema di Tagging e Categorizzazione è essenziale per rendere più semplice e intuitiva l'analisi delle transazioni da parte dell'utente.

La gerarchia tra categorie e tag è una struttura ad albero di un livello, con le categorie come radice e i tag come foglie, ad esempio:



Ogni transazione ha uno o più tag, e questi tag saranno appartenenti alla stessa categoria. In questo modo una transazione farà riferimento solamente ad una singola categoria

2.6 Statistica e Analisi visiva dei movimenti/bilancio

La sezione statistica è presente nella tab *Dashboard*

La sezione iniziale della statistica contiene il bilancio totale del gruppo ad oggi, il totale delle entrate e delle spese nel periodo selezionato. Questa prima parte serve a far capire in modo veloce e intuitivo la situazione nel periodo di tempo selezionato

Successivamente è presente un grafico in cui sono presenti le entrate e le uscite di quel periodo con le rispettive date

Infine abbiamo la statistica delle categorie in modo da far capire all'utente su quali categorie ci sono la maggior parte di entrate o uscite

Queste statistiche sono ricavate dai dati ricevuti dal modello, in questo modo sono sempre aggiornati e possono essere rimodellati per ulteriori interfacce grafiche

2.7 Filtraggio

Il meccanismo di filtraggio consente di filtrare i dati che vengono ricevuti attraverso: data, account, tags.

Attraverso il sistema di filtraggio, il model manda alla view solamente i dati necessari richiesti dall'utente.

Il sistema di filtraggio è stato implementato attravero l'interfaccia `FilterManager`, in modo da poter aggiungere nuove estensioni per implementare nuovi sistemi di filtraggio

2.8 Sistema di Fetching

Il sistema di fetching è stato implementato in modo che vengono estratti solamente i dati selezionati dal sistema di filtraggio.

Una volta estratti i dati, quest'ultimi vengono salvati nel `FetchManager` in modo da poter essere recuperati dall'interfaccia grafica

Il sistema di fetching sfrutta la programmazione concorrente, in modo da eseguire i vari fetch parallelamente e solamente dopo la fine di tutti i fetch viene eseguito l'aggiornamento dell'interfaccia grafica

3 Responsabilità Individuate

L'architettura del sistema è stata organizzata seguendo il pattern MVC (Model-View-Controller), con un'ulteriore suddivisione in componenti specializzati per gestire aspetti specifici dell'applicazione. Oltre alle classiche responsabilità di MVC, sono state individuate tre aree funzionali aggiuntive: Persistenza dei dati, Gestione dei Filtri e Fetching delle informazioni.

3.1 Responsabilità del Model

Il modello rappresenta il cuore dell'applicazione, incaricato di gestire tutte le entità finanziarie e le relative relazioni. Le principali componenti del dominio includono:

- FinancialGroup
- FinancialAccount
- FinancialCategory
- FinancialTag
- FinancialTransaction

- FinancialSchedule

Il modello si interfaccia con il layer di persistenza per garantire la coerenza e l'integrità dei dati, mantenendo al contempo una completa indipendenza dalle modalità specifiche di storage.

3.2 Responsabilità della View

La componente View rappresenta lo strato di presentazione del sistema, incaricato di gestire tutte le interazioni con l'utente finale. La sua architettura è stata progettata per garantire un'esperienza utente fluida e reattiva, mantenendo al contempo una netta separazione dalla logica di business.

Le sue responsabilità sono principalmente:

- Visualizzazione dei dati - Tramite aggiornamento dinamico dell'interfaccia in risposta a cambiamenti
- Gestione delle interazioni utente - Rilevamento input, validazione preliminare dei dati inseriti e feedback visivo
- Coordinamento con il Controller - Trasmissione degli eventi utente al Controller, ricezione aggiornamenti di stato

Ogni vista principale è implementata come componente indipendente, mantenendo un design uniforme

3.3 Responsabilità del Controller

Il Controller rappresenta il regista dell'applicazione, orchestrando il flusso di informazioni tra la View e il Model. Questo componente cruciale agisce come intermediario intelligente, trasformando le azioni dell'utente in operazioni concrete sul modello dati:

- Gestione degli eventi utente - Decide quali operazioni eseguire in base al contesto
- Coordinamento dati - recupera le informazioni richieste dal Model e prepara i dati per la visualizzazione
- Business logic applicativa - Applica le regole di business prima di modificare il Model

Il Controller garantisce che ogni azione utente venga tradotta correttamente in operazioni sul modello, mantenendo la coerenza dei dati e applicando tutte le regole di business necessarie, senza però conoscere i dettagli di implementazione né della visualizzazione né della persistenza dei dati.

3.4 Responsabilità della Persistenza

Il layer di Persistenza costituisce il fondamento dell'applicazione, garantendo l'integrità e la disponibilità dei dati finanziari nel tempo. Implementando il pattern Data Access Object (DAO), questo componente astrae completamente i dettagli di storage dal resto del sistema.

- Implementazione CRUD - Transazioni atomiche per garantire la persistenza, ottimizzazione query
- Astrazione del database - Utilizzo di Hibernate tramite JPA
- Gestione avanzata dei dati - Lazy loading per entità correlate, validazione integrità dei vincoli di dati

L'interfaccia CrudDAO fornisce un contratto chiaro per tutte le operazioni di persistenza, mentre l'implementazione concreta sfrutta al massimo le capacità di Hibernate per gestire relazioni complesse tra entità finanziarie, ereditarietà di tipi di transazione, e query avanzate per le funzionalità di reporting.

3.5 Responsabilità del Filtro

Il sistema di filtraggio opera in stretta sinergia con il componente di fetching per fornire un accesso efficiente e mirato ai dati finanziari. Questo meccanismo combinato garantisce che solo le informazioni rilevanti vengano caricate e processate, ottimizzando così le prestazioni dell'applicazione.

- Esecuzione fetching filtrato - La componente fetching riceve i filtri attivi

Questa implementazione garantisce che l'utente possa sempre accedere alla vista più pertinente dei propri dati finanziari, mantenendo al contempo un'impronta leggera sulle risorse di sistema e una chiara separazione delle responsabilità rispetto agli altri componenti MVC.

3.6 Responsabilità del Fetching

Il componente Fetching rappresenta lo strato di accesso ai dati intelligente dell'applicazione, orchestrando in modo ottimizzato il recupero e la preparazione delle informazioni finanziarie prima della presentazione all'utente. Funge da intermediario strategico tra i bisogni di visualizzazione e le sorgenti dati.

- Gestione delle richieste dati
- Preparazione dei dati

Questo componente trasforma radicalmente l'accesso ai dati da semplice operazione CRUD a processo intelligente e contestuale, diventando il vero differenziatore prestazionale dell'applicazione. La sua architettura aperta e modulare garantisce non solo eccellenti performance attuali, ma anche una solida base per integrare future innovazioni nel data management.

4 Classi e Interfacce Sviluppate

4.1 Modello

Le classi e le interfacce del livello modello rappresentano la base di tutto il sistema

4.1.1 Entità

`Interface Group` Definisce il contratto di tutte le entità di tipo Group. Estende `Taggable`, `Categorizable` e `Nameable`

`Abstract class AbstractGroup` Identifica gli attributi in comune tra tutte le entità di tipo Group. Contiene un GroupID, name e currency

`Class FinancialGroup` Identifica l'entità FinancialGroup. Contiene tags, accounts e categories

`Interface Account` Definisce il contratto di tutte le entità di tipo Account. Estende `Nameable`

`Abstract Class Abstract account` Identifica gli attributi in comune tra tutte le entità di tipo Account. Contiene accountID, name e initialAmount

`Class FinancialAccount` Identifica l'entità FinancialAccount. Contiene groupId, transactions, schedules

`Interface Category` Definisce il contratto di tutte le entità di tipo Category. Estende `Taggable` e `Nameable`

`Abstract Class AbstractCategory` Identifica gli attributi in comune tra tutte le entità di tipo Category. Contiene categoryId e name

`Class FinancialCategory` Identifica l'entità FinancialCategory. Contiene tags e groupId

`Interface Tag` Definisce il contratto di tutte le entità di tipo Tag. Estende `Nameable`

`Abstract Class AbstractTag` Identifica gli attributi in comune tra tutte le entità di tipo Tag. Contiene tagId e name

`Class FinancialTag` Identifica l'entità FinancialCategory. Contiene category, groupId, transactions e schedules

`Interface Transaction` Definisce il contratto di tutte le entità di tipo Trans-

action. Estende `Taggable`, `AccountLinked` e `Describable`

`Abstract Class AbstractTransaction` Identifica gli attributi in comune tra tutte le entità di tipo Transaction. Contiene transactionId, date, description, amount

`Class FinancialTransaction` Identifica l'entità FinancialTransaction. Contiene account e tags

`Class FinancialSchedule` Identifica l'entità FinancialSchedule. Contiene account e tags

`Interface Amount` Definisce il contratto di tutte gli embeddable di tipo Amount.

`Class FinancialAmount` Identifica l'embeddable FinancialAmount. Contiene amount

4.1.2 DAO

`Interface CrudDAO` Definisce il contratto di tutte le classi DAO

`Abstract Class AbstractDAO` Identifica gli attributi e i metodi in comune tra tutti i dao. Contiene entityClass

`Class GeneralDAO` Identifica un DAO generale per qualsiasi entità

`Class TransactionDAO` Identifica un DAO per l'entità Transaction

`Class CriteriaQueryHelper` Identifica un helper per la creazione di query

`Record TransactionUtil` Identifica una classe di utility per la creazione di transazioni verso il database

4.2 Controller

`Interface ServiceManager` Definisce il contratto di tutti i controller (Service Manager)

`Abstract Class AbstractServiceManager` Identifica gli attributi in comune tra tutti i service manager. Contiene changes, fetchManager, filterManager e daoManager

`Class FinancialServiceManager` Identifica un controller FinancialServiceManager

4.2.1 DAOManager

`Interface DaoManager` Definisce il contratto di tutti i daoManager. Estende RecurrenceTransaction

`Abstract Class AbstractDaoManager` Identifica gli attributi in comune tra tutti i dao manager. Contiene transactionDAO, accountDAO, categoryDAO, groupDAO, tagDAO, scheduleDAO

`Class FinancialDaoManager` Identifica un dao manager Financial

`Enumeration Recurrence` Identifica l'enumerazione recurrence

`Interface RecurrenceTransaction` Identifica il contratto per la generazione di transasazioni ricorrenti

4.2.2 FetchManager

`Interface FetchManager` Identifica il contratto di tutti i FetchManager. Estende UpdateGroup

`Abstract Class AbstractFetchManager` Identifica gli attributi in comune tra tutti i FetchManager

`Class FinancialFetchManager` Identifica un FinancialFetchManager

`Interface UpdateGroup` Identifica il contratto per aggiornare un gruppo

4.2.3 FilterManager

`Interface AccountsFilter` Identifica il contratto per implemtentare il filtraggio di account

`Interface DateFilter` Identifica il contratto per implementare il filtraggio tramite date

`Interface GroupFilter` Identifica il contratto per implementare il filtraggio di gruppi

`Interface ScheduleFilter` Identifica il contratto per implementare il selettore di schedule

`Interface TagsFilter` Identifica il contratto per implementare il filtraggio di tags

`Interface TransactionSelector` Identifica il contratto per implementare il selettore di transaction

`Interface FilterManager` Identifica il contratto per tutti i FilterManager

`AbstractClass AbstractFilterManager` Identifica gli attributi in comune tra tutti i filter manager. Contiene startDate, endDate, accounts, tags, group, transaction, schedule

`Class FinancialFilterManager` Identifica un FinancialFilterManager

4.3 View

L'interfaccia utente è stata sviluppata utilizzando JavaFX e FXML, una scelta che garantisce modularità e facilità di manutenzione. Grazie all'uso di un approccio dichiarativo (FXML) per la definizione del layout, la View risulta completamente disaccoppiata dalla logica di business, seguendo rigorosamente i principi del pattern MVC.

4.4 Interfacce generali

`Interface AccountLinked` Definisce il contratto per linkare un account ad un oggetto

`Interface Categorizable` Definisce il contratto per categorizzare un oggetto

`Interface Describable` Definisce il contratto per descrivere un oggetto

`Interface Nameable` Definisce il contratto per nominare un oggetto

`Interface Taggable` Definisce il contratto per taggare un oggetto

5 Organizzazione Dati e Persistenza

5.1 Organizzazione dei Dati

5.1.1 FinancialGroup

Un gruppo finanziario è costituito da:

- Un identificatore univoco `BigDecimal`
- Una currency `varchar`
- Un nome `varchar`

5.1.2 FinancialAccount

Un account finanziario è costituito da:

- Un identificatore univoco `bigint`
- Un initial amount `numeric`
- Un nome `varchar`
- Un gruppo associato `FinancialGroup`

5.1.3 FinancialCategory

Una categoria finanziaria è costituita da:

- Un identificatore univoco `bigint`
- Un nome `varchar`
- Un gruppo associato `FinancialGroup`

5.1.4 FinancialTag

Un tag finanziario è costituito da:

- Un identificatore univoco `bigint`
- Un nome `varchar`
- Una cateogoria associata `FinancialCategory`
- Un gruppo associato `FinancialGroup`

5.1.5 FinancialTransaction

Una transazione finanziaria è composta da:

- Un identificatore univoco `bigint`
- Una descrizione `varchar`
- Una data `date`
- Un ammountare `numeric`
- Un account associato `FinancialAccount`

5.1.6 FinancialSchedule

Una scandenza è composta da:

- Un identificatore univoco `bigint`
- Una descrizione `varchar`
- Una data `date`
- Un ammountare `numeric`
- Un account associato `FinancialAccount`

5.1.7 Relazione tag transazioni

La relazione tag-transazioni è composta da:

- Un identificativo della transazione `bigint`
- Un identificativo del tag `bigint`

5.1.8 Relazione tag scadenze

- Un identificativo della scadenza `bigint`
- Un identificativo del tag `bigint`

5.2 Persistenza

L'architettura di persistenza è stata progettata per garantire affidabilità, scalabilità e facilità di manutenzione, sfruttando le potenzialità di PostgreSQL come database relazionale e Hibernate JPA come strato di astrazione. Questa combinazione offre un equilibrio ottimale tra flessibilità e prestazioni.

Hibernate con JPA fornisce un'astrazione object-relational mapping (ORM) che permette di lavorare con oggetti Java anziché query SQL dirette. Le query JPA/Hibernate sono indipendenti dal dialetto SQL specifico, permettendo potenziali migrazioni verso altri RDBMS (MySQL, Oracle) con modifiche minime.

Criteria Builder: Utilizzato per costruire query dinamiche in modo type-safe, eliminando la necessità di stringhe SQL hardcoded e facilitando refactoring.

La soluzione di persistenza implementata non solo soddisfa i requisiti attuali, ma fornisce anche una base solida per evoluzioni future, mantenendo un equilibrio tra astrazione (JPA) e controllo (PostgreSQL avanzato)

6 Meccanismi Estendibilità

6.1 Estendibilità dell'interfaccia

L'architettura del sistema è stata concepita per garantire una completa indipendenza tra l'interfaccia utente e gli altri componenti dell'applicazione. Questo approccio strategico offre una flessibilità senza precedenti nell'adattamento a diverse piattaforme e tecnologie di presentazione.

Grazie alla rigorosa separazione tra View e Controller, ottenuta attraverso l' implementazione del pattern MVC, la componente grafica può essere riprogettata o sostituita senza alcun impatto sulla logica di business o sulla gestione dei dati. La comunicazione tra interfaccia e backend avviene esclusivamente attraverso un insieme ben definito di eventi e contratti di dati, mantenendo un

accoppiamento minimo ma sufficiente.

L'uso di JavaFX e FXML nell'implementazione corrente rappresenta solo una delle possibili scelte tecnologiche. La struttura modulare permette di sostituire questa tecnologia con qualsiasi altro framework di presentazione, purché rispetti i contratti stabiliti con il Controller. In particolare, la View è completamente agnostica rispetto ai meccanismi di persistenza e alle regole di business, delegando queste responsabilità agli altri componenti del sistema.

6.2 Estendibilità dei Filtri

Il sistema di filtraggio è stato progettato seguendo il principio Open/Closed, risultando contemporaneamente stabile nelle sue funzionalità di base e aperto a estensioni specializzate. L'interfaccia FilterManager funge da contratto fondamentale che garantisce la coerenza del sistema mentre permette l'introduzione di criteri di selezione sempre più sofisticati.

L'architettura a plug-in consente di aggiungere nuovi tipi di filtro senza modificare il codice esistente, semplicemente implementando nuove classi che aderiscono all'interfaccia stabilita. Ciascun filtro può così introdurre la propria logica specifica - come algoritmi di ricerca avanzata, criteri temporali complessi o analisi predittiva - mantenendo una perfetta integrazione con il sistema preesistente.

6.3 Estensione a Sistema di Gestione Risorse

L'architettura del progetto è stata concepita con un alto livello di astrazione, rendendo possibile la riconversione della soluzione da gestore di budget a sistema di gestione risorse con modifiche minime. Le interfacce e classi astratte alla base del design definiscono operazioni generiche per la tracciabilità e l'allocazione, indipendenti dal dominio specifico.

Questa flessibilità permette di riutilizzare il nucleo dell'applicazione in contesti diversi, come la gestione di magazzino, prenotazioni, o risorse aziendali. Ad esempio, le stesse meccaniche di filtraggio e analisi applicate alle transazioni finanziarie possono essere adattate per monitorare movimenti di merci, stati delle scorte, o utilizzo di attrezzature.

L'approccio basato su interfacce garantisce che l'estensione richieda solo l' implementazione di nuove entità (es. Product, Warehouse) e l'adeguamento delle regole di business, senza alterare la logica di persistenza, visualizzazione o controllo. Ciò riduce drasticamente lo sforzo di sviluppo per nuovi casi d'uso, dimostrando la versatilità del design originale.

6.4 Estendibilità della Persistenza

Il sistema di persistenza è stato progettato attorno a un'astrazione ben definita, che consente di modificare la tecnologia sottostante senza impattare il resto dell'applicazione. Grazie all'isolamento delle operazioni di accesso ai dati, è possibile sostituire l'implementazione corrente (PostgreSQL + Hibernate) con altre soluzioni, come database NoSQL, file strutturati (JSON/XML), o servizi cloud, semplicemente fornendo una nuova implementazione dell'interfaccia CrudDAO e FinancialTransactionDAO.

L'approccio adottato non solo semplifica eventuali migrazioni future, ma facilita anche il testing, poiché è possibile utilizzare repository in-memory per verificare il comportamento del sistema in ambienti isolati.