JBudget

Nicola Capancioni
Matricola: 122631

2025

Indice

1 Introduzione Progetto

La progettazione è basata sui principi dell'object-oriented programmingin particolare incapsulamento, ereditarietà e polimorfismo, con l'obiettivo di ottenere un'architettura robusta e facilmente estendibile. Il codice è organizzato secondo una separazione a livelli (Domain, Application, Infrastructure, Presentation), coerente con l'approccio Model-View-Controller: il modello e la logica di dominio risiedono nelle entità JPA e nei relativi servizi, la vista è gestita tramite controller JavaFX e la persistenza è incapsulata dietro interfacce repository. Tale separazione delle responsabilità rende il sistema più manutenibile, agevola il testing e permette la sostituzione o la modifica dei componenti con un basso impatto sull'intero progetto.

La scelta delle tecnologie - Java 21 come linguaggio principale, Gradle per la gestione delle dipendenze, JavaFX 17.0.2 per l'interfaccia grafica e Hibernate con H2 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 stayo 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 soldida base che può essere estesa in futiro per arricchire il sistema.

- Dashboard per l'analisi finanziaria
- Gestione dei movimenti e delle transazioni
- Gestione dei budget
- Gestione delle spese future e in scadenza

L'applicazione combina facilità d'uso e funzionalità avanzate: l'interfaccia intuitiva rende accessibili le operazioni di base, mentre gli strumenti specializzati soddisfano le esigenze di controllo dettagliato del 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
- Modularità e riusabilità dei componenti

2 Funzionalità Implementate

L'applicazione JBudget consente il monitoraggio e l'organizzazione delle spese domestiche e individuali. Al centro del sistema c'è una solida gestione dei movimenti finanziari, che permette di registrare, modificare ed eliminare ogni transazione con la possibilità di assegnarle categorie per una classificazione dettagliata. I movimenti possono essere filtrati per tipo (entrate o uscite) e per periodo temporale personalizzabile, offrendo una visualizzazione mirata dei dati. Il sistema di budget permette di pianifi-

care le spese per periodo e categoria, con complete funzionalità di creazione, modifica ed eliminazione dei budget esistenti, calcolo automatico dei valori reali basati sui movimenti effettuati e monitoraggio delle varianze tra pianificato e realizzato. Particolarmente utile risulta il modulo di scadenzario, che

gestisce le spese programmate e ricorrenti con operazioni complete di aggiunta, modifica ed eliminazione. Il sistema include filtri temporali predefiniti per visualizzare le scadenze dell'ultimo mese, degli ultimi 3 mesi, degli ultimi 6 mesi o dell'ultimo anno, automatizzando la creazione di movimenti per pagamenti periodici come utenze, a o rate. Il sistema di ricorrenze supporta frequenze giornaliere, settimanali, mensili e annuali, con gestione delle date di scadenza. Il modulo categorie include la

logica per la gestione completa delle classificazioni, consentendo di creare, modificare ed eliminare le categorie utilizzate per organizzare i movimenti finanziari. La logica è stata implementata, ma non ancora completata, costituendo così un possibile sviluppo per la seconda versione del progetto. Per

quanto riguarda l'analisi dei dati, la dashboard centrale genera statistiche e visualizzazioni immediate dell'andamento finanziario, con calcolo automatico di totali per categoria e trend mensili. Un sistema di filtri avanzato permette di analizzare specifici periodi temporali (ulimi 1, 3, 6, 12 mesi), offrendo una visione personalizzata della situazione economica grazie un grafico plot-line. La sincronizzazione

automatica tra movimenti e budget garantisce che i valori pianificati siano sempre confrontati con quelli reali, fornendo un quadro aggiornato delle performance finanziarie. Tutte queste funzionalità lavorano insieme per trasformare dati grezzi in informazioni utili per decisioni finanziarie consapevoli, con un'interfaccia che bilancia semplicità d'uso quotidiano e strumenti avanzati per analisi approfondite, garantendo il controllo completo sui dati attraverso operazioni di gestione integrate in ogni modulo dell'applicazione.

2.1 Gestione Movimenti

La gestione dei movimenti costituisce il cuore dell'applicazione di budget familiare, permettendo agli utenti di registrare e monitorare tutte le transazioni finanziarie attraverso un sistema completo e intuitivo. Ogni movimento viene caratterizzato da informazioni essenziali quali data, descrizione, tipo (entrata o uscita), importo e categoria di appartenenza. Il sistema supporta una classificazione flessibile attraverso un sistema di tag che consente di organizzare i movimenti in categorie specifiche come "Stipendio", "Utenze", "Alimentari", "Salute", "Svago" e "Trasporti".

2.1.1 Creazione Movimento

Per la creazione di un nuovo movimento bisogna cliccare il pulsante "Nuovo Movimento", al quale seguirà l'apertura di un dialog con campi obbligatori (descrizione, importo, tipo di transazione tramite dropdown "EXPENSE/INCOME", categoria) e opzionali (data e note). La conferma avviene tramite il pulsante "Salva".

2.1.2 Modifica Movimento

Selezionando un movimento dalla tabella e cliccando il pulsante "Modifica", si aprirà una dialog con i campi pre-compilati che permette di aggiornare qualsiasi informazione già inserita precedentemente. Il sistema mantiene l'integrità referenziale aggiornando automaticamente calcoli e statistiche correlate.

2.1.3 Elimina Movimento

Selezionando un movimento dalla tabella e cliccando il pulsante "Elimina", il sistema rimuove definitivamente il record dal database e aggiorna in tempo reale la visualizzazione tabellare.

2.1.4 Filtri di Ricerca dei Movimenti

Il sistema offre strumenti di filtro avanzati per visualizzare selettivamente i movimenti. Gli utenti possono filtrare per tipologia tramite dropdown ("INCOME", "EXPENSE" o entrambi), definire range temporali specifici attraverso campi data (dal/al), e utilizzare la ricerca testuale per descrizioni. Tutti

i filtri si integrano dinamicamente con la visualizzazione tabellare, aggiornando i risultati in tempo reale per facilitare analisi mirate e il reperimento rapido delle transazioni.

2.2 Gestione Scadenzario

Il modulo scadenzario consente di programmare spese ricorrenti e una tantum, offrendo una panoramica completa degli impegni finanziari futuri. L'interfaccia principale presenta una dashboard con statistiche aggregate che includono il totale attivo delle spese programmate, il conteggio delle spese scadute, quelle in scadenza oggi e nella settimana corrente. Il sistema categorizza visivamente le spese attraverso sezioni colorate: "Spese Scadute" (evidenziate in rosso), "In Scadenza Oggi" (in giallo) e "Prossima Settimana" (in verde), facilitando l'identificazione immediata delle priorità finanziarie.

2.2.1 Creazione Nuove Spese Programmate

L'utente può creare una nuova spesa programmata tramite il pulsante "Nuova Spesa" che apre un dialog dettagliato nel quale si richiedono di inserire campi obbligatori quali descrizione, importo, tipo (Uscita/Entrata), data di scadenza e ricorrenza. Il sistema supporta diversi tipi di ricorrenza attraverso un menu dropdown (Giornaliera, Settimanale, Mensile, Annuale, o "Nessuna" per spese una tantum). Per le spese ricorrenti, è possibile specificare l'intervallo in giorni e opzionalmente una data di fine ricorrenza. Il sistema permette inoltre di associare categorie multiple tramite una lista di selezione che include opzioni come "Alimentari", "Salute", "Stipendio" e altre categorie predefinite.

2.2.2 Modifica delle Spese Programmate

Selezionando una spesa dalla tabella principale e cliccando il pulsante "Modifica", si accede a un dialog identico a quello di creazione ma con tutti i campi pre-popolati con i dati correnti. L'utente può modificare qualsiasi parametro della spesa programmata, inclusa la ricorrenza e le categorie associate. Il sistema mantiene la coerenza dei dati aggiornando automaticamente le statistiche e le previsioni future.

2.2.3 Eliminazione e Completamento

Le spese possono essere rimosse definitivamente tramite il pulsante "Elimina" o marcate come completate con il pulsante "Completa". Quest'ultima funzionalità è particolarmente importante per il flusso di integrazione con il sistema dei movimenti.

2.2.4 Integrazione Automatica con i Movimenti

Il sistema implementa un meccanismo di integrazione automatica tra spese programmate e movimenti effettivi che rappresenta una delle caratteristiche più avanzate dell'applicazione. Quando una spesa programmata viene marcata come "Completa" o raggiunge automaticamente la sua data di scadenza, il sistema genera automaticamente un movimento corrispondente nella sezione "Movimenti" con tutti i dati della spesa originale (descrizione, importo, tipo, categoria, data). Questo processo garantisce che tutte le transazioni pianificate vengano correttamente registrate nel registro finanziario principale senza richiedere doppia immissione da parte dell'utente. Per le spese ricorrenti, il sistema crea un nuovo movimento ad ogni occorrenza della ricorrenza, aggiornando automaticamente la data della prossima scadenza e mantenendo attiva la spesa programmata fino alla data di fine ricorrenza (se specificata) o indefinitamente. Questo meccanismo assicura la tracciabilità completa delle spese periodiche come stipendi, affitti, utenze e altre spese fisse.

2.2.5 Aggiornamento Dinamico del Budget

L'integrazione tra scadenzario e movimenti garantisce l'aggiornamento automatico e in tempo reale di tutte le statistiche e analisi di budget. Quando i movimenti vengono generati dalle spese completate, il sistema ricalcola automaticamente tutti i totali, le proiezioni e le analisi comparative per categoria e periodo temporale. Questo approccio elimina le discrepanze tra budget pianificato e spese effettive, fornendo agli utenti una visione sempre aggiornata e accurata della loro situazione finanziaria.

Questa integrazione bidirezionale rappresenta un elemento centrale dell'architettura dell'applicazione, garantendo coerenza e affidabilità in tutti i moduli del sistema di gestione finanziaria familiare.

2.2.6 Filtri di Ricerca dello Scadenzario

Il sistema scadenzario offre strumenti di filtro avanzati per facilitare la navigazione e l'analisi delle spese programmate. Gli utenti possono filtrare le voci per stato tramite il dropdown "Solo attive", permettendo di visualizzare tutte le spese, solo quelle attive, in scadenza, già scadute, completate o ricorrenti. Il filtro per tipo di ricorrenza consente di selezionare spese specifiche (giornaliere, settimanali, mensili, annuali o una tantum). Inoltre, è disponibile un selettore di date con calendario interattivo che permette di filtrare le spese per periodi temporali specifici, facilitando la pianificazione e il monitoraggio delle scadenze. Questi strumenti di filtro si integrano dinamicamente con la visualizzazione tabellare, aggiornando i risultati in tempo reale e permettendo analisi mirate delle diverse tipologie di spese programmate.

2.3 Gestione Budget

Il sistema di budget costituisce il modulo di pianificazione finanziaria che permette di definire e monitorare obiettivi di entrate e uscite per periodi specifici e categorie predefinite, offrendo un controllo granulare sulla gestione del bilancio familiare. Il sistema organizza la pianificazione finanziaria attraverso una struttura matriciale che combina periodi temporali (mensili/annuali) e categorie di spesa (Utenze, Alimentari, Salute, Svago, Trasporti). Per ogni combinazione periodo-categoria, l'utente può definire valori pianificati di entrate e uscite che costituiscono il budget di riferimento. Il sistema confronta automaticamente questi valori con i dati reali provenienti dai movimenti effettivi, calcolando le varianze e identificando gli scostamenti dal piano originale. La dashboard principale presenta quattro sezioni informative codificate cromaticamente: Entrate Pianificate (azzurro), Entrate Reali (viola), Varianza Totale (verde quando positiva, rossa quando negativa) e Budget Sforati (rosso con conteggio numerico). Questa visualizzazione immediata permette una valutazione rapida delle performance finanziarie rispetto agli obiettivi prestabiliti.

2.3.1 Creazione Nuovo Budget

L'utente accede alla funzione tramite il pulsante "Nuovo Budget" che apre un dialog per la definizione dei parametri. Il form richiede la selezione del periodo temporale tramite dropdown, la categoria di riferimento (selezionabile tra quelle disponibili), e l'inserimento degli importi pianificati sia per entrate che per uscite. È possibile aggiungere note esplicative per documentare le assunzioni o particolarità del budget creato.

2.3.2 Modifica Budget Esistente

Selezionando una voce dalla tabella principale e utilizzando il pulsante "Modifica", si accede ai dati del budget selezionato con tutti i campi pre-popolati. L'utente può aggiornare qualsiasi parametro, inclusi i valori pianificati e le note associate. Il sistema ricalcola automaticamente le varianze in base ai nuovi parametri impostati.

2.3.3 Eliminazione Budget

Tramite il pulsante "Elimina", è possibile rimuovere definizioni di budget non più necessarie. Il sistema mantiene però uno storico dei dati reali già registrati per garantire la continuità delle analisi storiche.

2.3.4 Monitoraggio e Analisi del Budget

Il sistema calcola automaticamente le varianze confrontando i valori pianificati con quelli reali derivanti dai movimenti registrati. La colonna "Varianza" mostra la differenza tra entrate e uscite effettive rispetto al piano, mentre la colonna "Status" indica visivamente se il budget è "Nei limiti" (verde) o "Sforato" (rosso). Il contatore "Budget Sforati" nella dashboard fornisce un indicatore immediato del numero di budget che hanno superato i limiti pianificati.

2.3.5 Integrazione con i Movimenti e Aggiornamneto Dinamico

Il sistema budget si integra automaticamente con il modulo movimenti, utilizzando i dati delle transazioni effettive per popolare automaticamente le colonne "Entrate Reali" e "Spese Reali". Quando vengono registrati nuovi movimenti (sia manualmente che tramite il completamento di spese programmate), il sistema identifica automaticamente la categoria e il periodo di appartenenza e aggiorna i corrispondenti valori reali nel budget. Questo meccanismo garantisce che le analisi di varianza siano sempre basate su dati aggiornati e accurati.

2.3.6 Filtri di ricerca dei Budget

Il sistema offre funzionalità di filtro attraverso dropdown per "Tutti i periodi" e "Tutte le categorie", permettendo analisi focalizzate su specifici segmenti temporali o tipologie di spesa. I pulsanti "Applica" e "Pulisci" facilitano la navigazione tra diverse viste dei dati. La funzione "Aggiorna Valori Reali" consente di forzare un riallineamento completo tra movimenti e budget in caso di necessità.

2.4 Dashboard di Analisi e Statistiche

La dashboard di analisi rappresenta il centro di controllo statistico dell'applicazione, offrendo una panoramica completa e visuale delle performance finanziarie attraverso indicatori chiave e rappresentazioni grafiche interattive. Il sistema permette di selezionare diversi periodi di analisi tramite dropdown (Mese Corrente, Ultimi 3 Mesi, Ultimi 6 Mesi, Anno Corrente) con aggiornamento dinamico tramite il pulsante "Aggiorna". La dashboard presenta quattro KPI principali: Entrate Totali, Spese Totali, Bilancio (calcolato automaticamente come differenza) e Spese Scadute (con conteggio numerico). La sezione grafica include un grafico a torta per la distribuzione delle spese per categoria con legenda colorata, e un grafico plot line per il trend mensile del bilancio che evidenzia l'andamento temporale delle finanze. Completano la dashboard le sezioni "Spese in Scadenza" e "Movimenti Recenti" che forniscono informazioni immediate sulle prossime scadenze e le transazioni più recenti.

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 a queste responsabilità architetturali fondamentali, il sistema integra diverse aree funzionali trasversali che garantiscono robustezza e manutenibilità: Gestione della Persistenza attraverso il pattern Repository con implementazioni JPA, Sistema di Filtri per ricerca e analisi dinamica dei dati, Sincronizzazione Automatica tra componenti per mantenere coerenza informativa, Auditing e Tracciabilità per controllo delle modifiche, e Gestione degli Errori con logging centralizzato. Questa organizzazione rispetta i principi SOLID e facilita l'estendibilità del sistema per future funzionalità e integrazioni multi-piattaforma.

3.1 Responsabilità del Model

Il package `domain.model` implementa il cuore del domain model seguendo i principi del Domain-Driven Design, contenendo le entità di business che rappresentano i concetti fondamentali del sistema finanziario:

- **Movement**: Entità principale che gestisce i movimenti finanziari (data, descrizione, importo, tipo) con supporto per categorizzazione multipla e relazioni con piani di ammortamento
- **Budget**: Rappresenta budget pianificati per combinazioni periodo/categoria, con calcoli automatici di varianze e percentuali di utilizzo
- **Category**: Implementa struttura gerarchica per categorie di spesa con relazioni parent-child, validazioni di cicli e operazioni di navigazione
- **ScheduledExpense**: Modella spese programmate con ricorrenze complesse (giornaliere, settimanali, mensili, annuali) e gestione automatica delle prossime occorrenze

- **Period**: Gestisce periodi temporali per analisi con validazioni di coerenza e operazioni di contenimento date
- **AmortizationPlan**: Implementa piani di ammortamento con calcoli finanziari automatici per rate, interessi e quote capitali
- **MovementType, RecurrenceType**: Enumerazioni tipizzate per garantire type-safety

Le entità implementano validation logic nei costruttori, business methods per calcoli dominio-specifici, e relazioni JPA ottimizzate con fetch strategies appropriate.

3.2 Responsabilità della View

Il layer di presentazione gestisce l'interfaccia utente attraverso una combinazione di file FXML per layout dichiarativo e componenti JavaFX garantisce un'esperienza utente fluida e reattiva, mantenendo al contempo una netta separazione dalla logica di business.

3.2.1 File FXML e Layout

- **main-view.fxml**: Layout principale con MenuBar, TabPane per navigazione modulare e BorderPane responsive
- **movements-view.fxml**: Interfaccia per gestione movimenti con TableView, filtri avanzati e form dialog
- **budgets-view.fxml**: Dashboard budget con tabelle comparative, KPI summary e controlli di aggiornamento
- **dashboard-view.fxml**: Centro di controllo con PieChart per categorie, LineChart per trend temporali e widget statistiche
- **scheduled-expenses-view.fxml**: Scadenzario con visualizzazione multi-stato e azioni rapide

3.2.2 Dialog Personalizzati

- **AddScheduledExpenseDialog**: Dialog complesso per spese programmate con validation real-time, gestione ricorrenze e selezione multipla categorie

3.3 Responsabilità del Controller

Il package `presentation.controller` rappresenta il coordinatore dell'applicazione seguendo il pattern MVC, con controller specializzati per ogni modulo funzionale:

3.3.1 Controller Principali

- **MainController**: Orchestratore principale che gestisce navigation tra tab, lifecycle dell'applicazione, dependency injection manuale dei servizi e comunicazione cross-controller
- **MovementController**: Gestisce CRUD movimenti con validation forms, filtri dinamici, bulk operations e sincronizzazione automatica budget
- **BudgetController**: Coordina operazioni budget con aggiornamento real-time valori effettivi, calcolo varianze e dashboard KPI
- **DashboardController**: Controller statistiche con gestione chart interattivi, selezione periodi dinamica e aggregazioni complesse
- **ScheduledExpenseController**: Gestisce scadenzario con stati multipli, operazioni batch completion e quick actions per scadenze
- **CategoryController**: Controlla gerarchia categorie con TreeView navigation e validazioni struttura ad albero

3.3.2 Pattern di Comunicazione

I controller implementano observer pattern per sincronizzazione automatica, command pattern per operazioni complesse, e notification system per aggiornamenti cross-modulari attraverso `MainController.refreshAllTabsFromExternal()`.

3.4 Gestione della Persistenza

Il layer di persistenza implementa il pattern Repository con tecnologia JPA/Hibernate, organizzato in due livelli per separare responsabilità di dominio da dettagli tecnici.

3.4.1 Repository Interfaces (domain.repository)

Le interfacce definiscono contratti per accesso dati senza vincolare implementazione:

- **MovementRepository**: Query specifiche per filtri temporali, categoriali, ricerca testuale e aggregazioni statistiche
- **BudgetRepository**: Metodi per recupero per periodo/categoria, identificazione budget sforati e ordinamento per varianze
- **CategoryRepository**: Operazioni gerarchia con validazioni parent-child, query ancestors/descendants
- **ScheduledExpenseRepository**: Query temporali complesse per scadenze, stati e ricorrenze
- **PeriodRepository**: Gestione sovrapposizioni, contenimento date e query temporali

3.4.2 Implementazioni JPA (infrastructure.persistence)

Le implementazioni concrete utilizzano Hibernate con ottimizzazioni specifiche:

- **JpaMovementRepository**: JOIN FETCH per relazioni ManyToMany, query native per statistiche, gestione transazionale con rollback
- **JpaBudgetRepository**: Query aggregate con calcoli real-time, JOIN complessi per varianze
- **JpaCategoryRepository**: Mapping gerarchie con strategie lazy/eager loading appropriate
- **JpaScheduledExpenseRepository**: Query temporali ottimizzate con indici su date

Le implementazioni includono extensive logging per debugging, exception handling con RuntimeException wrapping, e transaction management esplicito per consistency.

3.5 Gestione dei Filtri

Il sistema di filtri è implementato trasversalmente su repository, service e controller layers per garantire flessibilità e performance:

3.5.1 Implementazione Repository Level

Repository implementano metodi specializzati utilizzando Criteria API JPA per costruzione dinamica query. Metodi come `findByDateRange()`, `findByCategoryAndType()`, `findByStatus()` permettono combinazioni multiple senza proliferazione metodi specifici.

3.5.2 Coordinamento Service Layer

Services orchestrano logica filtro complessa, combinando criteri multipli e gestendo cache risultati per performance. Forniscono metodi come `getFilteredMovements(FilterCriteria criteria)` astraendo complessità da controller.

3.5.3 Interfaccia Controller Layer

Controller JavaFX gestiscono UI per filtri catturando input (dropdown, date picker, search fields) e traducendoli in parametri servizi. Implementano pattern Observer per aggiornamento dinamico viste quando filtri cambiano.

3.6 Fetching delle Informazioni

Il fetching segue approccio stratificato separando responsabilità tra coordinamento, trasformazione dati e logica business.

3.6.1 Application Layer - Coordinamento

DTO Classes: Implementano Data Transfer Object pattern per disaccoppiamento layers:

- **BudgetDTO**: Contiene metodi helper che garantiscono la sicurezza contro i valori null, assicurando robustezza e affidabilità durante la manipolazione e la presentazione dei dati di budget all'inter
- **MovementDTO**: Continene tutti i metadati necessari per la visualizzazione ottimale dei movimenti, includendo funzionalità per la mappatura delle categorie e utilità per la formattazione dei dati secondo le esigenze dell'interfaccia utente.
- **StatisticsDTO**: Incorpora metodi di validazione integrati che garantiscono l'integrità e la coerenza dei dati statistici prima della loro presentazione, assicurando che le informazioni visualizzate siano sempre accurate e affidabili per l'analisi del budget familiare.

Service Layer: Coordinano operazioni busines, implementando una validazione estensiva dei dati e garantendo la sincronizzazione tra le diverse entità del sistema:

- **MovementServiceImpl**: Funge da orchestratore principale per la gestione dei movimenti finanziari, implementando la sincronizzazione automatica con i budget associati, l'inizializzazione intelligente delle categorie e un sistema completo di logging per tracciare tutte le operazioni eseguite.
- **BudgetServiceImpl**: Si occupa della gestione di calcoli complessi che coinvolgono periodi temporali e categorie di spesa, mantenendo automaticamente aggiornati i valori reali in base ai movimenti effettuati e garantendo la coerenza dei dati di budget.
- **ScheduledExpenseServiceImpl**: Coordina la gestione delle spese ricorrenti attraverso un'analisi avanzata dei pattern temporali, permettendo di prevedere e pianificare automaticamente le future occorrenze delle spese programmate.
- **StatisticsServiceImpl**: Aggregatore sofisticato che raccoglie dati da multiple fonti per produrre analisi delle tendenze e statistiche comparative, fornendo insights dettagliati sull'andamento del budget familiare nel tempo.

Use Cases: Gli Use Case fungono da orchestratori per le operazioni che richiedono il coordinamento di servizi multipli:

- **AddMovementUseCase**: Coordina l'intero processo di aggiunta di un nuovo movimento finanziario, gestendo la validazione dei dati in ingresso, l'aggiornamento automatico dei budget correlati e l'invio delle notifiche necessarie per mantenere sincronizzata l'interfaccia utente.
- **CreateBudgetUseCase**: Si occupa della creazione di nuovi budget implementando controlli di unicità per evitare duplicazioni e popolando automaticamente i valori predefiniti secondo le configurazioni del sistema, garantendo coerenza e completezza dei dati.
- **GenerateStatisticsUseCase**: Esegue aggregazioni complesse attingendo da fonti dati multiple, implementando una validazione rigorosa delle date e dei periodi di riferimento per assicurare l'accuratezza delle statistiche generate.
- **SyncDataUseCase**: Rappresenta un framework estendibile progettato specificamente per supportare future implementazioni di sincronizzazione multi-dispositivo, fornendo le basi architetturali per l'espansione delle funzionalità di condivisione dati.

3.7 Infrastructure Configuration

3.7.1 ApplicationConfig

La configurazione dell'applicazione implementa un sistema di iniezione manuale delle dipendenze basato sul pattern Singleton con inizializzazione lazy. Questo approccio risolve intelligentemente le dipendenze circolari attraverso un ordinamento strategico delle fasi di inizializzazione e fornisce metodi dedicati per il testing dell'integrazione tra servizi, garantendo un avvio pulito e controllato dell'applicazione.

3.7.2 HibernateConfig

La configurazione di Hibernate utilizza il meccanismo del double-checked locking per garantire la thread safety in ambienti concorrenti. Le configurazioni vengono caricate dinamicamente da file properties per offrire massima flessibilità durante il deployment, mentre l'integrazione con HikariCP assicura un connection pooling efficiente e performante per l'accesso al database.

3.7.3 DatabaseConfig

Il setup del database si basa su H2 embedded con creazione automatica dello schema e script di inizializzazione per il seeding dei dati di base. Questa configurazione permette un avvio immediato dell'applicazione senza dipendenze esterne, facilitando sia lo sviluppo che il testing dell'applicazione.

4 Estendibilità e Manutenibilità

4.1 Architettura Modulare

La separazione netta tra i diversi livelli architetturali facilita significativamente lo sviluppo di nuove estensioni e funzionalità. Nuovi moduli possono essere integrati nell'applicazione senza compromettere o impattare la funzionalità esistente, grazie alla chiara definizione delle interfacce e delle responsabilità di ciascun componente. Questa struttura modulare permette agli sviluppatori di lavorare in parallelo su diverse parti del sistema mantenendo l'integrità complessiva dell'applicazione.

4.2 Gestione delle Configurazioni

Il sistema di configurazione basato su file properties offre una flessibilità eccezionale nel cambio di ambiente, permettendo di passare facilmente tra configurazioni di sviluppo, test e produzione senza modificare il codice sorgente. Il supporto per le migrazioni del database garantisce un'evoluzione controllata dello schema dati, facilitando gli aggiornamenti futuri e la manutenzione del sistema nel tempo.

4.3 Preparazione per API

L'architettura del service layer è stata progettata specificamente per facilitare una futura esposizione tramite REST API, rendendo possibile l'integrazione con applicazioni web o mobile senza dover ristrutturare la logica di business esistente. Il pattern DTO è già predisposto per la serializzazione JSON, garantendo una transizione fluida verso architetture distribuite e permettendo l'interoperabilità con sistemi esterni.

5 Classi e Interfacce Sviluppate

Il sistema JBudget è strutturato attraverso un'architettura a layer che separa chiaramente le responsabilità tra dominio di business, logica applicativa, persistenza e presentazione. Questa sezione descrive nel dettaglio l'organizzazione delle classi e interfacce sviluppate, evidenziando le relazioni tra componenti e i pattern architetturali implementati.

5.1 Entità del Dominio (Domain Model)

5.1.1 Movement

La classe `Movement` rappresenta l'entità core per i movimenti finanziari, implementando attraverso annotazioni JPA una mappatura complessa verso il database relazionale. La classe gestisce relazioni ManyToMany con le categorie attraverso una tabella di join `movement_categories`, mentre mantiene una relazione opzionale con piani di ammortamento per movimenti derivati da rate. L'entità implementa metodi di business per validazione importi, gestione categorie associate e calcolo di importi con segno basati sul tipo di movimento.

Attributi principali:

- `Long id`: Identificativo univoco con strategia IDENTITY
- `String description`: Descrizione del movimento con validazione not-null
- `BigDecimal amount`: Importo con precisione 19,2 per calcoli finanziari accurati
- `MovementType type`: Enumerazione per tipo movimento (INCOME/EXPENSE)
- `LocalDate date`: Data del movimento con validazione obbligatoria
- `Set<Category> categories`: Relazione ManyToMany con fetch EAGER per performance
- `LocalDateTime createdAt, updatedAt`: Audit timestamps automatici

Metodi di business significativi:

- `addCategory(Category)`: Gestione associazioni categoriali con logging
- `getSignedAmount()`: Calcolo importo con segno basato sul tipo
- `isIncome()/isExpense()`: Metodi di convenience per controllo tipo
- `updateDetails()`: Aggiornamento completo dati con audit automatico

5.1.2 Budget

L'entità `Budget` implementa il modello di pianificazione finanziaria attraverso una combinazione unica di periodo e categoria, garantita da un vincolo di unicità a livello database. La classe incorpora logiche sofisticate per il calcolo automatico di varianze, percentuali di utilizzo e identificazione di situazioni di sforamento budget.

Attributi specializzati:

- `Period period`: Relazione ManyToOne con fetch LAZY per ottimizzazione
- `Category category`: Relazione opzionale (null per budget generali)
- `BigDecimal plannedIncome/plannedExpenses`: Valori pianificati
- `BigDecimal actualIncome/actualExpenses`: Valori effettivi calcolati
- `boolean active`: Flag per soft-delete e filtraggio

Metodi di calcolo:

- `getVarianceBalance()`: Calcolo varianza tra bilancio effettivo e pianificato
- `getIncomePercentage()/getExpensesPercentage()`: Percentuali di realizzazione
- `isOverBudget()`: Identificazione sforamenti con logica business
- `updateActuals()`: Aggiornamento valori effettivi con debug logging

5.1.3 ScheduledExpense

La classe `ScheduledExpense` modella la complessità delle spese programmate con pattern temporali articolati, implementando algoritmi sofisticati per il calcolo delle prossime occorrenze e la gestione degli stati del ciclo di vita.

Gestione ricorrenze:

- `RecurrenceType recurrenceType`: Enumerazione per tipi ricorrenza
- `Integer recurrenceInterval`: Intervallo personalizzabile (ogni N unità)
- `LocalDate recurrenceEndDate`: Data fine ricorrenza opzionale
- `getNextDueDate()`: Algoritmo calcolo prossima scadenza
- `createNextOccurrence()`: Generazione automatica prossima istanza

Stati e transizioni:

- `boolean completed/active`: Flags per gestione stati
- `isDue()/isOverdue()`: Controlli stato temporale
- `getDaysUntilDue()`: Calcolo giorni rimanenti
- `createMovement()`: Conversione in movimento effettivo

5.1.4 Category

L'entità `Category` implementa una struttura gerarchica complessa attraverso relazioni self-referencing, con validazioni sofisticate per prevenire cicli e garantire integrità strutturale. È importante sottolineare che la logica per la struttura gerarchica complessa è stata solo iniziata in questa versione dell'applicazione, rappresentando un'importante area di sviluppo prevista per la seconda release del sistema.

Struttura gerarchica:

- `Category parent`: Relazione ManyToOne auto-referenziante
- `Set<Category> children`: OneToMany per navigazione discendenti
- `isRoot()/isLeaf()`: Metodi identificazione posizione gerarchica
- `isDescendantOf()`: Validazione relazioni gerarchiche
- `getPath()`: Calcolo percorso completo dalla radice
- `getAllDescendants()`: Recupero ricorsivo tutti discendenti

5.1.5 AmortizationPlan

La classe `AmortizationPlan` implementa calcoli finanziari complessi per piani di ammortamento, utilizzando algoritmi matematici precisi per rate, interessi e quote capitali.

Calcoli finanziari:

- `generateInstallments()`: Algoritmo generazione rate complete
- `calculateMonthlyPayment()`: Calcolo rata mensile con formula finanziaria
- `getTotalInterest()`: Calcolo interessi totali pagati
- `getCompletedInstallments()`: Monitoraggio progresso pagamenti
- `getRemainingAmount()`: Calcolo debito residuo

5.2 Interfacce Repository

5.2.1 MovementRepository

L'interfaccia `MovementRepository` definisce un contratto completo per l'accesso ai dati dei movimenti finanziari, incorporando query specializzate per ogni tipologia di filtro e aggregazione richiesta dal dominio finanziario. L'interfaccia è strutturata per coprire tutti i pattern di accesso ai dati necessari alla gestione del budget familiare, dalle operazioni CRUD di base fino alle aggregazioni statistiche più complesse. Il design dell'interfaccia privilegia l'efficienza delle query attraverso metodi ottimizzati che riducono il numero di accessi al database e supportano operazioni di calcolo direttamente a livello di persistenza, garantendo prestazioni adeguate anche con volumi consistenti di dati storici.

Query temporali e categoriali:

- `findByDateBetween()`: Filtro range temporale ottimizzato
- `findByCategory()/findByCategoriesContaining()`: Query categoriali
- `findByType()`: Filtro per tipo movimento
- `findByPeriod()`: Query per periodo specifico

Aggregazioni statistiche:

- `getTotalByTypeAndDateRange()`: Totali per tipo e periodo
- `getTotalByCategoryAndDateRange()`: Aggregazioni per categoria
- `findByDescriptionContaining()`: Ricerca testuale
- `findAllPaginated()`: Supporto paginazione per performance

5.2.2 BudgetRepository

L'interfaccia `BudgetRepository` specializza l'accesso ai dati relativi ai budget attraverso un insieme di query ottimizzate specificamente progettate per supportare analisi comparative approfondite e l'identificazione tempestiva di anomalie finanziarie. Questa interfaccia si concentra sulla gestione delle relazioni complesse tra periodi temporali e categorie di spesa, fornendo metodi dedicati per il recupero e l'analisi dei dati di pianificazione finanziaria. Le query implementate permettono di effettuare confronti dettagliati tra valori pianificati e realizzati, facilitando il monitoraggio delle performance del budget e l'individuazione di scostamenti significativi che richiedono attenzione da parte dell'utente.

Query specializzate:

- `findByPeriodAndCategory()`: Query per combinazione unica
- `findOverBudgets()`: Identificazione budget sforati
- `findByPeriodOrderByVariance()`: Ordinamento per performance
- `findGeneralBudgets()`: Query per budget senza categoria

5.2.3 ScheduledExpenseRepository

L'interfaccia `ScheduledExpenseRepository` incorpora un sistema di query temporali progettate per la gestione avanzata delle scadenze e il monitoraggio degli stati delle spese programmate. Questa interfaccia gestisce la complessità dei pattern ricorrenti e delle logiche temporali, fornendo metodi specializzati per identificare spese scadute, imminenti o in stato di completamento. Le query implementate supportano algoritmi di calcolo delle ricorrenze e permettono un controllo granulare sui diversi stati del ciclo di vita delle spese programmate, facilitando la pianificazione finanziaria e la gestione proattiva delle scadenze future.

Query temporali avanzate:

- `findOverdueExpenses()`: Spese scadute non completate
- `findDueToday()/findDueThisWeek()`: Scadenze immediate
- `findByRecurrenceType()`: Filtro per tipo ricorrenza
- `findRecurringExpenses()`: Tutte le spese ricorrenti

5.3 Implementazioni JPA

5.3.1 JpaMovementRepository

L'implementazione di `JpaMovementRepository` utilizza Hibernate Session implementando ottimizzazioni per la gestione efficiente delle relazioni ManyToMany e l'esecuzione di query aggregate complesse. La classe adotta strategie di JOIN FETCH mirate per risolvere i problemi di performance N+1 tipici delle relazioni tra movimenti e categorie, garantendo il caricamento dei dati correlati in una singola query.

Ottimizzazioni implementate:

- JOIN FETCH per categorie con SELECT DISTINCT
- Query native per aggregazioni performance-critical
- Transaction management con exception handling
- Logging esteso per debugging e monitoraggio

5.4 Service Layer

5.4.1 MovementServiceImpl

Il servizio `MovementServiceImpl` implementa logiche di coordinamento tra movimenti e budget, orchestrando automaticamente la sincronizzazione dei dati finanziari ogni volta che viene eseguita un'operazione sui movimenti. Il servizio gestisce l'inizializzazione automatica delle categorie predefinite attraverso un sistema di lazy loading che garantisce la presenza delle categorie essenziali al primo utilizzo dell'applicazione. Una caratteristica distintiva di questa implementazione è la sincronizzazione cross-modulo che mantiene automaticamente allineati i valori dei budget con i movimenti effettivi, aggiornando in tempo reale le statistiche e notificando gli altri componenti dell'interfaccia utente dei cambiamenti. Il servizio incorpora inoltre un sistema di validazione comprensivo con messaggi di errore personalizzati e logging esteso per tracciare tutte le operazioni critiche.

Funzionalità avanzate:

- Automatic budget synchronization dopo ogni operazione CRUD
- Lazy initialization categorie predefinite con pattern Singleton
- Comprehensive validation con custom exception messages
- Cross-controller notification per UI updates

5.4.2 BudgetServiceImpl

5.4.3 BudgetServiceImpl

Il servizio `BudgetServiceImpl` orchestra calcoli per l'aggiornamento automatico dei valori reali dei budget basandosi sui movimenti effettivamente registrati nel sistema. L'implementazione gestisce intelligentemente la distribuzione delle entrate su tutti i budget del periodo di riferimento, mentre le spese vengono associate specificamente alle categorie corrispondenti attraverso algoritmi di matching sofisticati. Il servizio implementa logiche avanzate per la gestione delle relazioni temporali tra periodi e categorie, con creazione automatica di nuove combinazioni periodo-categoria quando necessario. I

calcoli di varianza vengono eseguiti in tempo reale con ottimizzazioni delle performance che minimizzano l'impatto computazionale, mentre un sistema di fallback garantisce la correttezza dei totali anche in presenza di categorie non specificate o budget generali.

Logiche di business sofisticate:

- Algoritmo distribuzione entrate su tutti i budget del periodo
- Filtro spese per categoria specifica con fallback a totali
- Automatic period/category creation con date range mapping
- Real-time variance calculation con performance optimization

5.4.4 StatisticsServiceImpl

Il servizio implementa aggregazioni multi-source per statistiche finanziarie, con trend analysis e comparative analytics tra periodi diversi.

Capacità analitiche:

- **Calcolo delle tendenze mensili**: Analisi dei trend con riempimento completo dei periodi per garantire continuità temporale nelle statistiche anche in presenza di mesi senza movimenti
- **Analisi delle performance di budget**: Valutazione delle prestazioni finanziarie attraverso indicatori chiave di performance (KPI) aggregati che forniscono una visione d'insieme dell'andamento del budget
- **Categorie di spesa principali**: Identificazione delle categorie con maggiore impatto economico attraverso classifiche dinamiche che si aggiornano automaticamente in base ai dati correnti
- **Confronto multi-periodo**: Analisi comparative tra diversi periodi temporali con calcolo delle varianze per identificare trend e anomalie nell'andamento finanziario

5.5 Controller JavaFX

5.5.1 MainController

Il controller principale `MainController` rappresenta il punto di coordinamento centrale dell'intera applicazione, implementando un sistema di dependency injection manuale che gestisce l'inizializzazione e il wiring di tutti i componenti del sistema. Questo controller orchestra il ciclo di vita completo dell'applicazione, dalla fase di bootstrap iniziale fino alla chiusura controllata, garantendo che tutte le risorse vengano correttamente allocate e rilasciate. Una delle caratteristiche distintive del MainController è l'implementazione di pattern di comunicazione sofisticati tra i diversi controller dell'interfaccia utente. Attraverso un sistema di eventi e notifiche, il controller principale facilita lo scambio di informazioni tra le diverse sezioni dell'applicazione senza creare dipendenze dirette, mantenendo così un'architettura pulita e modulare. Il controller gestisce inoltre la navigazione dell'interfaccia utente attraverso un sistema di tab dinamiche, creando e inizializzando i controller specifici solo quando necessario, ottimizzando così l'utilizzo delle risorse e i tempi di avvio dell'applicazione.
Responsabilità architetturali:

- **Inizializzazione dei servizi**: Gestione dell'avvio dei servizi con controllo dell'ordine delle dipendenze per garantire che tutti i componenti siano correttamente inizializzati secondo le loro interdipendenze
- **Navigazione basata su tab**: Sistema di navigazione a schede con creazione dinamica dei controller, permettendo un'interfaccia modulare e un caricamento ottimizzato delle sezioni dell'applicazione
- **Comunicazione cross-controller**: Implementazione del pattern Observer per facilitare la comunicazione tra diversi controller senza creare dipendenze dirette, mantenendo il disaccoppiamento architetturale

- **Gestione del ciclo di vita dell'applicazione**: Controllo completo del ciclo di vita con pulizia automatica delle risorse durante la chiusura per prevenire memory leak e garantire un shutdown pulito

5.5.2 MovementController

5.5.3 MovementController

Il controller `MovementController` gestisce un'interfaccia utente per la gestione dei movimenti finanziari, implementando un sistema di validazione in tempo reale che fornisce feedback immediato all'utente durante l'inserimento dei dati. Il controller supporta operazioni bulk per la gestione efficiente di multipli movimenti simultaneamente e incorpora un sistema di filtraggio avanzato che permette ricerche sofisticate basate su criteri temporali, categoriali e di importo. L'interfaccia utilizza cell factory personalizzate per la formattazione monetaria e il color coding dei diversi tipi di movimento, mentre il property binding dinamico gestisce automaticamente gli stati dei pulsanti in base alla validità dei dati inseriti. Il controller implementa inoltre menu contestuali con operazioni specifiche per azione e mantiene la sincronizzazione automatica con i budget attraverso notifiche cross-controller dopo ogni modifica dei dati.
Caratteristiche UI avanzate:

- **Cell factory personalizzate**: Implementazione di formattatori monetari specifici e codifica a colori per differenziare visivamente i tipi di movimento e facilitare la lettura dei dati finanziari
- **Validazione dinamica**: Sistema di validazione in tempo reale con binding delle proprietà che gestisce automaticamente gli stati dei pulsanti in base alla validità dei dati inseriti
- **Menu contestuali**: Menu di scelta rapida con operazioni specifiche per ogni azione, permettendo accesso immediato alle funzionalità più utilizzate
- **Notifiche automatiche di budget**: Sistema di notifica che aggiorna automaticamente i budget correlati dopo ogni modifica dei dati dei movimenti

5.6 Data Transfer Objects

5.6.1 MovementDTO, BudgetDTO, StatisticsDTO

Le classi DTO implementano null-safety patterns con helper methods, comprehensive validation e format utilities per presentation layer optimization. Ogni DTO include metodi di calcolo derivati, validazione business rules e conversion utilities per seamless integration tra layer architetturali.
Questa organizzazione in classi e interfacce riflette un'architettura matura che separa chiaramente le responsabilità, facilita il testing attraverso dependency injection, e garantisce maintainability attraverso principi SOLID ben implementati. La struttura supporta estensibilità futura e migration verso architetture distribuite mantenendo la coerenza del domain model.

6 Limitazioni Attuali

L'applicazione presenta alcune limitazioni che caratterizzano questa prima versione del sistema:

- **Applicazione desktop mono-utente**: Il sistema è progettato per un singolo utente alla volta, senza supporto per accessi concorrenti o gestione multi-utente
- **Database H2 con accesso limitato**: L'utilizzo del database embedded H2 limita le capacità di accesso concorrente e non supporta scenari di utilizzo distribuito
- **Capacità di reporting limitate**: Le funzionalità di generazione report sono basilari e non coprono tutte le esigenze di analisi finanziaria avanzata
- **Assenza di funzionalità import/export**: Il sistema non supporta attualmente l'importazione o l'esportazione di dati da/verso formati esterni o altri sistemi di gestione finanziaria

7 Sviluppi Futuri

L'architettura modulare dell'applicazione è stata progettata per supportare numerosi miglioramenti e estensioni future:

- **Supporto multi-utente con autenticazione**: Implementazione di un sistema completo di gestione utenti con autenticazione sicura e controllo degli accessi per supportare famiglie o gruppi di utenti
- **Interfaccia web con Spring Boot**: Sviluppo di una versione web dell'applicazione utilizzando Spring Boot per garantire accessibilità da qualsiasi dispositivo e browser
- **Reporting avanzato con JasperReports**: Integrazione di un sistema di reporting professionale per generare analisi dettagliate e report personalizzabili
- **Servizi di sincronizzazione cloud**: Implementazione di servizi cloud per la sincronizzazione automatica dei dati tra dispositivi multipli e backup sicuro
- **Integrazione con API bancarie**: Connessione diretta con i servizi bancari per l'importazione automatica dei movimenti e la riconciliazione dei conti
- **Completamento della struttura gerarchica dei tag/categorie**: Finalizzazione dell'implementazione della gestione gerarchica avanzata delle categorie con validazioni complete per cicli e ottimizzazioni delle performance per navigazione multi-livello