



Spendly: Documentazione Tecnica

Amin Borqal, Loris Iacoban, Diego Rossi

2 marzo 2025

Indice

1 Iterazione 0	6
1.1 Introduzione	6
1.2 Requisiti	8
1.3 Casi d'Uso	9
1.4 Architettura del Sistema	12
1.4.1 Model	12
1.4.2 View	13
1.4.3 Controller	13
1.4.4 Flusso delle Richieste	13
1.5 Priorità casi d'uso	14
1.6 Topologia del Sistema	16
1.6.1 Protocollo HTTP e REST-API	16
1.6.2 Caratteristiche dell'architettura	16
1.7 Strumenti Utilizzati	18
2 Iterazione 1	21
2.1 Casi d'Uso Implementati	21
2.1.1 UC1: Login	21
2.1.2 UC2: Registrazione	21
2.1.3 UC3: Logout	22
2.1.4 UC7: Creazione Gruppo	22
2.1.5 UC8: Invita Membri	23
2.1.6 UC9: Elimina Membri	23
2.1.7 UC10: Modifica Membri	23
2.1.8 UC11: Eliminazione Gruppo	24
2.1.9 UC12: Accesso a un Gruppo	24
2.2 Diagramma delle Interfacce: Utente e Gruppo	25
2.3 UML Class Diagram	26
2.4 UML Component Diagram	28
2.5 UML Deployment Diagram	29
2.6 Testing	31
2.6.1 Analisi Statica - CodeMR	31
2.6.2 Analisi Dinamica - JUnit	33
2.6.3 API Esposte	35

3 Iterazione 2	41
3.1 Casi d'Uso Implementati	41
3.1.1 UC4: Creazione Alert di Gruppo	41
3.1.2 UC6: Eliminazione Alert	42
3.1.3 UC13: Inserimento Spesa	42
3.1.4 UC14: Eliminazione Spesa	42
3.1.5 UC15: Modifica Spesa	43
3.1.6 UC16: Scegli Tipologia Spesa	43
3.1.7 UC17: Visualizzazione Spese	44
3.1.8 UC18: Ricalcolo Debiti	44
3.2 Diagramma delle Interfacce: Spesa	45
3.3 UML Class Diagram	46
3.4 UML Component Diagram	49
3.5 UML Deployment Diagram	50
3.6 Testing	51
3.6.1 Analisi Statica - CodeMR	51
3.6.2 Analisi Dinamica - JUnit	53
3.6.3 API Costi	55
3.7 Algoritmo ottimizza debiti	63
3.7.1 PseudoCodice	63
3.7.2 Descrizione	65
3.7.3 Complessità	66
4 Iterazione 3	67
4.1 Introduzione	67
4.2 Casi d'Uso	67
4.2.1 Casi d'uso Implementati	68
4.2.2 UC20: Inserimento Spesa Personale	68
4.2.3 UC21: Eliminazione Spesa Personale	69
4.2.4 UC22: Modifica Spesa Personale	69
4.2.5 UC23: Scegli Tipologia Spesa Personale	69
4.2.6 UC24: Pagamento Spesa Personale	70
4.2.7 UC25: Visualizzazione Spesa Personale	70
4.2.8 UC26: Aggiunta Denaro a Budget	70
4.2.9 UC27: Togli Denaro da Budget	71
4.2.10 UC28: Visualizzazione Budget	71
4.2.11 UC29: Creazione Risparmio	71
4.2.12 UC30: Visualizzazione Risparmio	72

4.2.13	UC31: Modifica Risparmio	72
4.2.14	UC32: Elimina Risparmio	72
4.3	Diagramma delle Interfacce	73
4.4	UML Class Diagram per tipi di dato	74
4.5	UML Component Diagram	75
4.6	UML Deployment Diagram	76
4.7	Testing	77
4.7.1	Analisi Statica - CodeMR	77
4.7.2	Analisi Dinamica - JUnit	79
4.7.3	API	81

Elenco delle figure

1	Diagramma casi d'uso	9
2	M-V-C Pattern	12
3	Diagramma dell'architettura della webapp Spendly	16
4	UML Deployment Diagram	30
5	Complexity	31
6	Coupling	31
7	Problemi classi	32
8	Problemi classi 2	32
9	Struttura dei package	33
10	Test per la classe Group	34
11	Test per la classe GroupService	34
12	Test per la classe GroupController	34
13	Risultato API Registrazione	36
14	Risultato API Login	37
15	Risultato API Creazione Gruppo	38
16	Risultato API Inserimento utente	39
17	Risultato API Visualizzazione Gruppi	40
18	UML Deployment Diagram	50
19	Complexity	51
20	Coupling	51
21	Problemi classi	52
22	Struttura dei package	52
23	Test per ottimizzare Debiti	53
24	Test aggiornati per la classe Group	53
25	Test aggiornati per la classe GroupService	54
26	Test aggiornati per la classe GroupController	54
27	Test per la classe CostService	55
28	Test per la classe CostController	55
29	Risultato API Aggiunta Costo	57
30	Risultato API Eliminazione Costo	57
31	Risultato API Visualizzazione Costi Utente	58
32	Risultato API Visualizzazione Costi Gruppo	59
33	Risultato API Creazione Alert	60
34	Risultato API eliminazione Alert	61
35	Risultato API Visualizzazione Alert di Gruppo	62
36	Funzione CreaDizionario	63

37	Funzione CalcolaDebiti	64
38	Diagramma casi d'uso	67
39	UML Deployment Diagram	75
40	Complexity	76
41	Coupling	76
42	Problemi classi	77
43	Problemi classi	77
44	Struttura dei package	78
45	Test per la classe SavingService	79
46	Test per la classe SavingController	79
47	Test per la classe BudgetService	80
48	Test per la classe BudgetController	80
49	API creazione budget	81
50	API visualizza budget di un utente	82
51	API per aggiungere fondi al budget	83
52	API per sottrarre fondi al budget	84
53	API per creare un piano di risparmio	86
54	API per ottenere i piani di risparmio dell'utente	87
55	API per aggiungere fondi al piano di risparmio dell'utente	88
56	API per togliere fondi al piano di risparmio dell'utente	89
57	API per eliminare un piano di risparmio	90

Elenco delle tabelle

1	Iterazione1	21
2	Iterazione2	41
3	Iterazione 3 - Casi d'Uso Implementati.	68

1 Iterazione 0

1.1 Introduzione

Spendly è una web app progettata per semplificare la gestione delle spese personali e di gruppo, offrendo funzionalità che permettono a singoli utenti, famiglie o gruppi di organizzarsi in modo efficiente.

Con Spendly, è possibile:

- Creare gruppi di utenti, dove ciascun membro può registrare le spese effettuate per conto degli altri. L'applicazione calcolerà automaticamente il numero minimo di transazioni necessarie per saldare i debiti all'interno del gruppo.
- Impostare avvisi personalizzati per monitorare il livello di spesa del gruppo, fornendo notifiche utili per evitare superamenti del budget condiviso.

A livello individuale, ogni utente può:

- Visualizzare, registrare e gestire le proprie spese in modo autonomo.
- Creare e monitorare un budget di risparmio per pianificare meglio le proprie finanze personali.

Grazie a queste funzionalità, Spendly rende la gestione economica più semplice, trasparente ed efficace, sia per singoli utenti che per gruppi.

Perché scegliere Spendly?

Spendly è strutturata attorno a una serie di funzionalità che coprono ogni aspetto della gestione delle spese. Queste funzionalità sono state sviluppate con l'obiettivo di offrire la massima flessibilità, sia per gli utenti individuali che per i gruppi di condivisione spese. Spendly è più di una semplice applicazione di gestione delle spese. È una soluzione completa che integra funzionalità avanzate, come il calcolo automatico dei debiti e la gestione centralizzata delle spese. Questo la rende ideale per ogni tipo di utente, dai gruppi di amici che vogliono semplificare la divisione delle spese, alle famiglie che desiderano tenere sotto controllo i propri budget. Con una piattaforma sicura, flessibile e facile da usare, Spendly trasforma il modo in cui le persone gestiscono i

loro soldi, riducendo lo stress finanziario e migliorando la trasparenza nelle relazioni economiche. Spendly è lo strumento perfetto per chi desidera una gestione semplice, efficace e organizzata delle spese. Con funzionalità pensate per rispondere alle esigenze di utenti individuali e gruppi, questa web-app rappresenta una soluzione innovativa e accessibile per migliorare la vita quotidiana di chiunque voglia un controllo totale sulle proprie finanze.

1.2 Requisiti

Il sistema deve soddisfare i seguenti requisiti funzionali e non funzionali:

- **Requisiti Funzionali:**

- Gli utenti devono poter registrarsi e autenticarsi al sistema.
- Gli utenti devono poter aggiungere, modificare e cancellare le proprie spese.
- Gli utenti devono poter creare gruppi e gestire le spese condivise.
- Il sistema deve calcolare automaticamente il bilancio dei debiti tra i membri del gruppo.
- Il sistema deve inviare notifiche o alert agli utenti quando il loro budget per una determinata categoria di spesa viene superato.
- Gli utenti devono poter impostare budget personalizzati per categorie specifiche (es. alimentari, trasporti, svago, ecc.).

- **Requisiti Non Funzionali:**

- L'applicazione deve essere intuitiva e facile da usare.
- Il sistema deve garantire la sicurezza dei dati personali degli utenti mediante crittografia e autenticazione sicura.
- Le transazioni e le modifiche devono essere sincronizzate in tempo reale tra tutti i dispositivi degli utenti.
- L'applicazione deve essere accessibile da diversi dispositivi (smartphone, tablet, desktop).
- Il sistema deve garantire elevate prestazioni, assicurando una risposta rapida alle richieste degli utenti.

1.3 Casi d'Uso

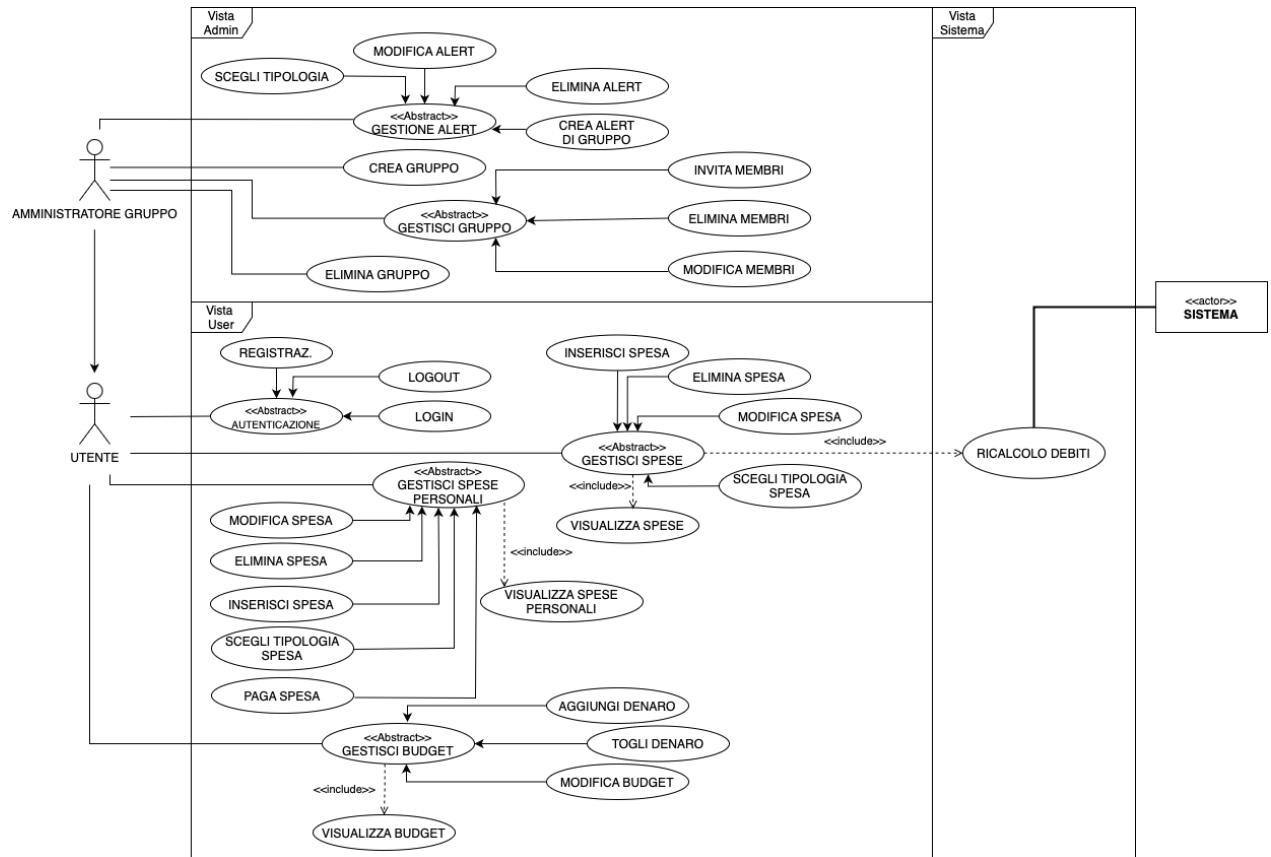


Figura 1: Diagramma casi d'uso

Di seguito sono riportati i casi d'uso in figura:

- **UC1 - Login:** Come utente, voglio poter accedere al mio account per gestire le mie spese.
- **UC2 - Registrazione:** Come nuovo utente, voglio potermi registrare al sistema per iniziare a usare la web-app.
- **UC3 - Logout:** Voglio poter terminare la mia sessione.
- **UC4 - Crea alert di gruppo:** Voglio poter inserire un alert per segnalare il raggiungimento di una soglia limite di spesa nel gruppo.

- **UC5 - Modifica alert:** Voglio poter modificare i valori di un alert.
- **UC6 - Elimina alert:** Voglio poter eliminare un alert precedentemente impostato.
- **UC7 - Scegli tipologia alert:** Voglio poter scegliere la tipologia di alert da applicare.
- **UC8 - Crea gruppo:** Voglio poter creare un gruppo di condivisione spese.
- **UC9 - Invita membri:** Voglio, in qualità di amministratore, invitare membri nel gruppo.
- **UC10 - Elimina membri:** Voglio poter eliminare membri dal gruppo.
- **UC11 - Modifica membri:** Voglio poter modificare i dettagli dei membri del gruppo.
- **UC12 - Elimina gruppo:** Voglio poter eliminare un gruppo di condivisione spese.
- **UC13 - Accedi gruppo:** Voglio poter accedere ad un gruppo di condivisione spese.
- **UC14 - Inserisci spesa:** Voglio poter inserire una spesa di gruppo.
- **UC15 - Elimina spesa:** Voglio poter eliminare una spesa di gruppo.
- **UC16 - Modifica spesa:** Voglio poter modificare una spesa di gruppo.
- **UC17 - Scegli tipologia spesa:** Voglio poter scegliere la tipologia di spesa.
- **UC18 - Visualizza spese:** Voglio poter visualizzare le spese di gruppo.
- **UC19 - Ricalcolo debiti:** Voglio poter calcolare i debiti tra i membri.
- **UC20 - Inserisci spesa personale:** Voglio poter inserire una spesa personale.

- **UC21 - Elimina spesa personale:** Voglio poter eliminare una spesa personale.
- **UC22 - Modifica spesa personale:** Voglio poter modificare una spesa personale.
- **UC23 - Scegli tipologia spesa personale:** Voglio poter scegliere la tipologia di spesa personale.
- **UC24 - Paga spesa personale:** Voglio poter saldare una spesa personale utilizzando i metodi di pagamento.
- **UC25 - Visualizza spesa personale:** Voglio poter visualizzare le spese personali.
- **UC26 - Aggiungi denaro a budget:** Voglio poter inserire un budget di risparmio.
- **UC27 - Togli denaro da budget:** Voglio poter eliminare un budget di risparmio.
- **UC28 - Visualizza budget:** Voglio poter visualizzare i budget di risparmio.

1.4 Architettura del Sistema

Il sistema segue il pattern architettonico **Model-View-Controller (MVC)**, un'architettura software che separa la logica di presentazione, la logica di business e la gestione dei dati. Questa separazione consente una maggiore modularità e facilita la manutenzione e l'estendibilità del sistema. Le tre componenti principali sono:

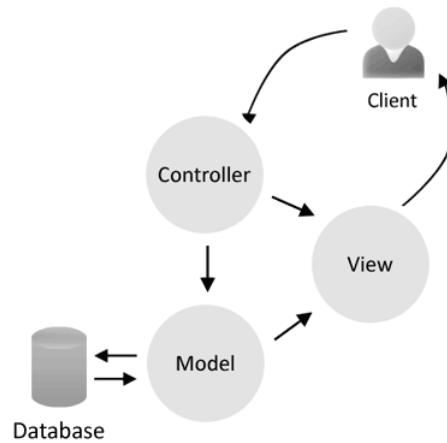


Figura 2: M-V-C Pattern

1.4.1 Model

Il **Model** rappresenta la logica di business e la gestione dei dati. In questo sistema:

- I dati sono memorizzati in un database relazionale **PostgreSQL**, scelto per la sua affidabilità e scalabilità.
- Il backend, implementato in **Spring Boot**, gestisce le operazioni sui dati attraverso un'architettura basata su servizi.
- I principali servizi includono:
 - **Servizio Utente**: gestione della registrazione, autenticazione (tramite **JWT**) e gestione del profilo utente.
 - **Servizio Gestione Spese**: esecuzione delle operazioni CRUD sulle spese degli utenti.

- **Servizio Gruppi:** creazione e gestione dei gruppi di utenti.

- L’interazione con il database avviene tramite il **Data Access Layer**, basato su JPA e Hibernate.

1.4.2 View

La **View** è responsabile della presentazione dei dati all’utente e della gestione dell’interazione con il sistema. Nel nostro caso:

- Il frontend è sviluppato utilizzando **Vue.js**, un framework JavaScript progressivo che permette lo sviluppo di interfacce utente reattive e modulari.
- Le chiamate API al backend vengono gestite tramite **Axios**, consentendo un flusso di dati asincrono tra client e server.
- L’autenticazione degli utenti è gestita tramite **JWT**, che vengono memorizzati e inviati nelle richieste HTTP per garantire la sicurezza.

1.4.3 Controller

Il **Controller** funge da intermediario tra il Model e la View, gestendo le richieste utente e applicando la logica di business. In questo sistema:

- Il backend espone un’API **RESTful** tramite Spring Boot, con endpoint per la gestione di utenti, spese e gruppi.
- Ogni richiesta HTTP viene elaborata da un controller dedicato, che interagisce con i servizi del Model per recuperare o aggiornare i dati.
- Le risposte sono restituite in formato JSON, consentendo una comunicazione fluida con il frontend.

1.4.4 Flusso delle Richieste

Il flusso di una tipica richiesta utente segue questi passi:

1. L’utente interagisce con il frontend Vue.js tramite un’interfaccia web.
2. Un’azione (es. login, aggiunta di una spesa) genera una richiesta HTTP verso il backend.

3. Il **Controller** riceve la richiesta e invoca il servizio appropriato.
4. Il **Model** esegue la logica di business e interagisce con il database per recuperare o aggiornare i dati.
5. La risposta viene inviata al frontend in formato JSON.
6. Il frontend aggiorna dinamicamente l'interfaccia utente con i nuovi dati.

Questa architettura garantisce una chiara separazione delle responsabilità, facilitando lo sviluppo, la scalabilità e la manutenzione del sistema.

1.5 Priorità casi d'uso

I casi d'uso sono suddivisi in tre categorie, in linea con l'ordine stabilito in **Casi d'Uso**:

ID	Titolo
UC1	Login
UC2	Registrazione
UC3	Logout
UC8	Crea gruppo
UC9	Invita membri
UC10	Modifica membri
UC11	Elimina gruppo
UC13	Accedi gruppo
UC14	Inserisci spesa
UC15	Elimina spesa
UC16	Modifica spesa
UC17	Scegli tipologia spesa
UC18	Visualizza spese
UC19	Ricalcolo debiti

Tabella 1: Coda alta priorità

ID	Titolo
UC4	Crea alert di gruppo
UC5	Modifica alert
UC6	Elimina alert
UC7	Scegli tipologia alert
UC20	Inserisci spesa personale
UC21	Elimina spesa personale
UC22	Modifica spesa personale
UC23	Scegli tipologia spesa personale
UC24	Paga spesa personale
UC25	Visualizza spesa personale

Tabella 2: Coda a media priorità

ID	Titolo
UC26	Aggiungi denaro a budget
UC27	Togli denaro da budget
UC28	Visualizza budget

Tabella 3: Coda a bassa priorità

Perché si dividono in code di priorità i casi d'uso?

In quanto lo sviluppo del progetto viene sviluppato tramite il processo Agile Model-Driven Development.

AMDD combina i principi della modellazione guidata dallo sviluppo con la flessibilità dei metodi agili, che a differenza dello sviluppo tradizionale, in cui tutta l'analisi viene completata prima di iniziare la programmazione, prevede una modellazione iniziale leggera e iterativa, concentrata solo sugli aspetti essenziali per l'iterazione corrente.

Ad ogni ciclo di sviluppo, il team seleziona un numero limitato di casi d'uso dalla coda di priorità più alta, lavorando su di essi fino a ottenere una funzionalità completa e testata. Questo approccio:

- Riduce il rischio affrontando le funzionalità critiche prima possibile.
- Favorisce il feedback continuo, permettendo agli stakeholder di validare il progresso e suggerire miglioramenti.

- Adatta lo sviluppo alle necessità reali, permettendo di rivalutare le priorità in base all'andamento del progetto.

Grazie a questo metodo, il sistema viene costruito in modo solido e incrementale, con la possibilità di aggiungere nuove funzionalità man mano che il progetto evolve.

1.6 Topologia del Sistema

La topologia del sistema è rappresentata nel seguente schema:



Figura 3: Diagramma dell'architettura della webapp Spendly

Il sistema è basato unicamente su un'architettura **web**, in cui i client interagiscono con un **web server** sviluppato in **Spring Boot**. I client utilizzano un browser, comunicando con il server attraverso **REST-API HTTP**, con dati in formato **JSON**.

1.6.1 Protocollo HTTP e REST-API

HTTP è un protocollo di trasferimento ipertestuale che offre un meccanismo semplice e universale per inviare e ricevere dati. È ideale per le nostre REST-API poiché assicura la compatibilità con i browser e l'interscambio di dati in JSON su qualsiasi piattaforma.

1.6.2 Caratteristiche dell'architettura

- **Scalabilità:** possibilità di scalare l'intero sistema o le singole funzionalità in base alle necessità.
- **Modularità:** sviluppo e manutenzione semplificati grazie all'organizzazione delle funzionalità in moduli indipendenti.

- **Affidabilità:** eventuali malfunzionamenti in una funzionalità non compromettono l'intero sistema.
- **Manutenibilità:** possibilità di aggiornare o sostituire singoli moduli senza interrompere l'operatività complessiva.

1.7 Strumenti Utilizzati

Lo sviluppo del progetto è supportato da una serie di strumenti software che facilitano la progettazione, lo sviluppo, il testing e la gestione del codice. Di seguito, una panoramica dettagliata dei tool impiegati:

- **Visual Studio Code:** IDE (Integrated Development Environment) scelto per lo sviluppo del progetto. VS Code offre un ambiente di sviluppo leggero ma potente, con un ampio supporto per il linguaggio Java e numerose estensioni utili. Tra i plugin utilizzati troviamo:
 - **Spring Boot Extension Pack:** per il supporto avanzato nello sviluppo di applicazioni Spring Boot.
 - **Java Extension Pack:** per il supporto completo al linguaggio Java, con funzionalità di autocompletamento, refactoring e debugging.
- **Spring Boot:** Framework Java basato su Spring, progettato per lo sviluppo di applicazioni web scalabili e strutturate secondo l'architettura a microservizi. Grazie alla sua configurazione automatica e al supporto integrato per REST API, Spring Boot semplifica la gestione del backend e garantisce un'elevata manutenibilità del codice.
- **PostgreSQL:** Sistema di gestione di database relazionale (RDBMS) scelto per la sua affidabilità, scalabilità e aderenza agli standard SQL. PostgreSQL supporta transazioni ACID (Atomicità, Coerenza, Isolamento, Durabilità) ed è ottimizzato per operazioni complesse e interrogazioni avanzate. Il database è stato configurato per garantire prestazioni elevate e sicurezza dei dati.
- **Postman:** Strumento essenziale per il testing delle API REST sviluppate con Spring Boot. Consente di effettuare richieste HTTP, validare le risposte del server e automatizzare test, facilitando così il debug e l'integrazione tra i diversi componenti del sistema.
- **Grok AI:** Tecnologia avanzata per la generazione automatica di immagini basata su intelligenza artificiale. Grok AI viene utilizzato per creare rappresentazioni visive intuitive e schematiche di concetti complessi, supportando la documentazione e la comunicazione grafica del progetto.

- **JUnit 4:** Framework per il testing unitario in Java, impiegato per validare il corretto funzionamento delle classi e dei metodi sviluppati. L'uso di test automatizzati consente di rilevare tempestivamente eventuali errori e di garantire la robustezza del codice.
- **JGraphT:** Libreria Java dedicata alla modellazione e alla gestione di strutture dati basate su grafi. Utilizzata per la rappresentazione e la manipolazione di relazioni complesse all'interno del sistema.
- **CodeMR:** Strumento avanzato per l'analisi della qualità del codice Java e la visualizzazione delle metriche software. CodeMR consente di valutare la complessità del codice, individuare problemi di design e migliorare la manutenibilità del progetto.
- **GitHub:** Piattaforma per il versionamento del codice basata su Git, utilizzata per il controllo delle versioni e la collaborazione tra gli sviluppatori. Grazie a GitHub, è possibile tracciare le modifiche al codice, gestire le revisioni e garantire un workflow ordinato ed efficiente.
- **GitHub Desktop:** Applicazione con interfaccia grafica che semplifica l'interazione con il repository GitHub direttamente dal PC. Permette di eseguire operazioni di commit, push e pull senza dover utilizzare la riga di comando, facilitando la gestione del codice per gli sviluppatori.
- **StarUML:** Software utilizzato per la modellazione di diagrammi UML (Unified Modeling Language), fondamentale nella fase di progettazione dell'architettura del sistema. StarUML consente di rappresentare visivamente classi, casi d'uso e flussi operativi.
- **diagrams.net:** Applicazione web per la creazione di diagrammi e schemi con notazione libera. Utilizzata per rappresentare graficamente flussi di dati, processi e architetture software, facilitando la comprensione e la condivisione delle informazioni progettuali.
- **WhatsApp:** Applicazione di messaggistica utilizzata come strumento di comunicazione interna tra i membri del team. Attraverso WhatsApp, è possibile coordinare le attività di sviluppo, discutere problemi tecnici e organizzare riunioni in tempo reale, garantendo un flusso comunicativo rapido ed efficiente.

- **Telegram:** Applicazione di messaggistica utilizzata dal team per discussioni tecniche e condivisione rapida di documenti, codice e aggiornamenti di progetto. Grazie ai suoi bot e alle funzionalità avanzate di gestione dei gruppi, Telegram rappresenta uno strumento utile per il coordinamento del lavoro.

2 Iterazione 1

2.1 Casi d'Uso Implementati

In questa iterazione sono stati sviluppati i seguenti casi d'uso ritenuti prioritari per lo sviluppo di **Spendly**.

ID	Titolo
UC1	Login
UC2	Registrazione
UC3	Logout
UC7	Crea gruppo
UC8	Invita membri
UC9	Elimina membri
UC10	Modifica membri
UC11	Elimina gruppo
UC12	Accedi gruppo

Tabella 4: Iterazione1

2.1.1 UC1: Login

Attori: Utente, Sistema.

Descrizione: L'utente può autenticarsi nel sistema per accedere al proprio account.

Flusso degli eventi:

1. L'utente accede alla pagina di login.
2. Inserisce email e password.
3. Il sistema verifica le credenziali e autentica l'utente.
4. L'utente viene reindirizzato alla dashboard.

2.1.2 UC2: Registrazione

Attori: Utente, Sistema.

Descrizione: Un nuovo utente può registrarsi creando un account.

Flusso degli eventi:

1. L'utente accede alla pagina di registrazione.
2. Inserisce nome, email e password.
3. Il sistema verifica che l'email non sia già registrata.
4. Se la verifica è superata, il sistema crea l'account e lo memorizza.
5. L'utente viene reindirizzato alla dashboard.

2.1.3 UC3: Logout

Attori: Utente, Sistema.

Descrizione: L'utente può terminare la sessione ed effettuare il logout.

Flusso degli eventi:

1. L'utente clicca su "Logout".
2. Il sistema invalida la sessione e mostra la schermata di login.

2.1.4 UC7: Creazione Gruppo

Attori: Utente amministratore, Sistema.

Descrizione: L'utente può creare un nuovo gruppo di spese per la condivisione con altri membri.

Flusso degli eventi:

1. L'utente clicca su "Crea Gruppo".
2. Inserisce il nome del gruppo e una descrizione opzionale.
3. Il sistema crea il gruppo e assegna l'utente come amministratore.
4. L'utente viene reindirizzato alla pagina del gruppo.

2.1.5 UC8: Invita Membri

Attori: Utente amministratore, Sistema.

Descrizione: L'amministratore di un gruppo può invitare altri utenti a unirsi al gruppo di spese.

Flusso degli eventi:

1. L'amministratore accede alla pagina del gruppo.
2. Clicca su "Invita Membri" e inserisce l'email degli utenti da invitare.
3. Il sistema invia un'email con l'invito e memorizza la richiesta.
4. Gli utenti ricevono l'invito e possono accettarlo per entrare nel gruppo.

2.1.6 UC9: Elimina Membri

Attori: Utente amministratore, Sistema.

Descrizione: L'amministratore di un gruppo può rimuovere un membro dal gruppo.

Flusso degli eventi:

1. L'amministratore accede alla lista dei membri del gruppo.
2. Seleziona il membro da rimuovere e clicca su "Elimina".
3. Il sistema rimuove il membro e aggiorna la lista.

2.1.7 UC10: Modifica Membri

Attori: Utente amministratore, Sistema.

Descrizione: L'amministratore di un gruppo può modificare i dettagli dei membri (ad esempio assegnare nuovi ruoli).

Flusso degli eventi:

1. L'amministratore accede alla lista dei membri.

2. Seleziona un membro e modifica i dettagli (es. ruolo nel gruppo).
3. Il sistema aggiorna i dati e notifica il cambiamento.

2.1.8 UC11: Eliminazione Gruppo

Attori: Utente amministratore, Sistema.

Descrizione: L'amministratore di un gruppo può eliminare definitivamente un gruppo di spese.

Flusso degli eventi:

1. L'amministratore accede alle impostazioni del gruppo.
2. Clicca su "Elimina Gruppo".
3. Il sistema chiede conferma prima di procedere.
4. Se confermato, il gruppo e tutte le sue spese vengono eliminate.

2.1.9 UC12: Accesso a un Gruppo

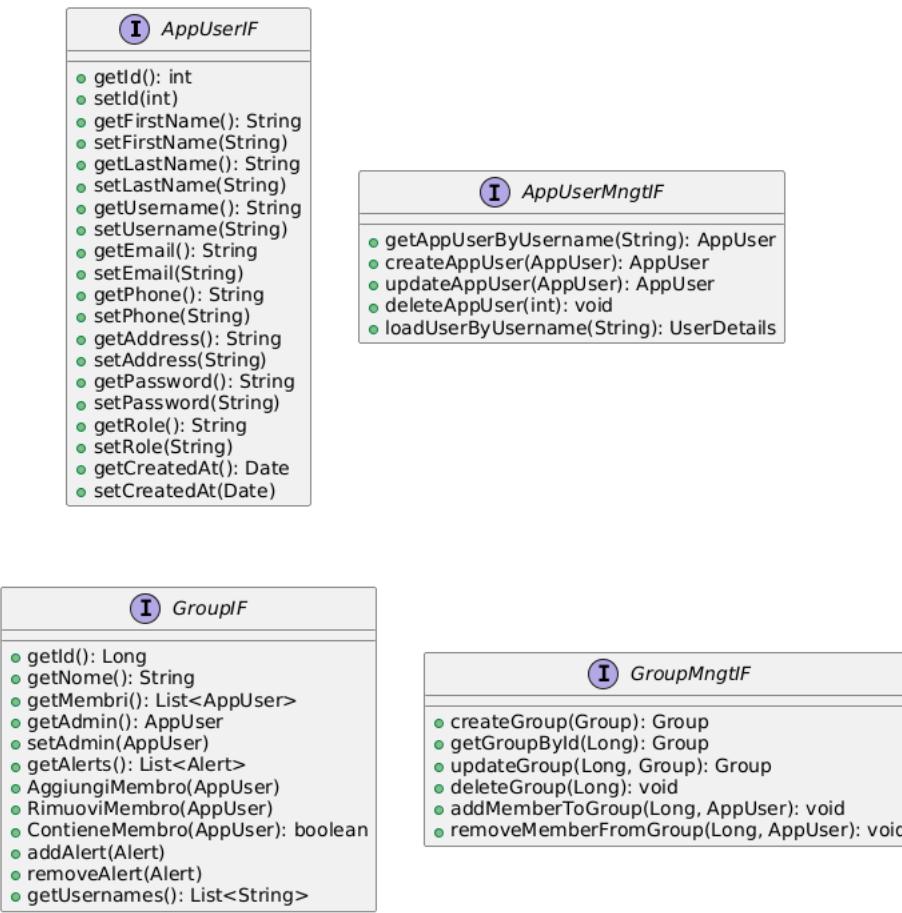
Attori: Utente, Sistema.

Descrizione: Un utente può accedere a un gruppo di spese a cui è stato invitato.

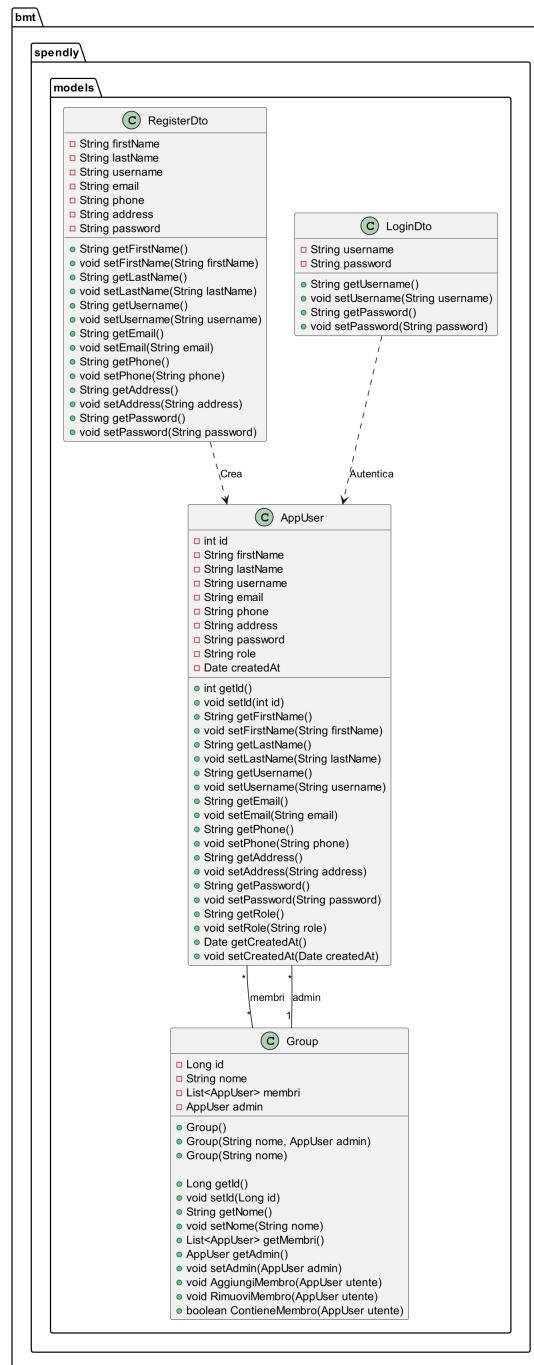
Flusso degli eventi:

1. L'utente riceve un invito via email o notifica nell'app.
2. Clicca sul link di invito e accede alla web-app.
3. Il sistema verifica la validità dell'invito e aggiunge l'utente al gruppo.
4. L'utente viene reindirizzato alla pagina del gruppo.

2.2 Diagramma delle Interfacce: Utente e Gruppo



2.3 UML Class Diagram



LoginDto → AppUser:

- Rappresenta il processo di autenticazione nel sistema.
- La relazione mostra che la classe **LoginDto** contiene i dati necessari (username e password) per verificare l'identità di un utente già registrato (**AppUser**).
- In altre parole, **LoginDto** fornisce un mezzo per confermare che un utente (**AppUser**) ha accesso al sistema.

RegisterDto → AppUser:

- Rappresenta il processo di registrazione di un nuovo utente.
- La relazione indica che la classe **RegisterDto** contiene i dati necessari (nome, cognome, email, password, ecc.) per creare un nuovo account utente (**AppUser**) all'interno del sistema.
- Questo significa che la classe **RegisterDto** viene utilizzata per tradurre i dati di registrazione in un'istanza valida della classe **AppUser**.

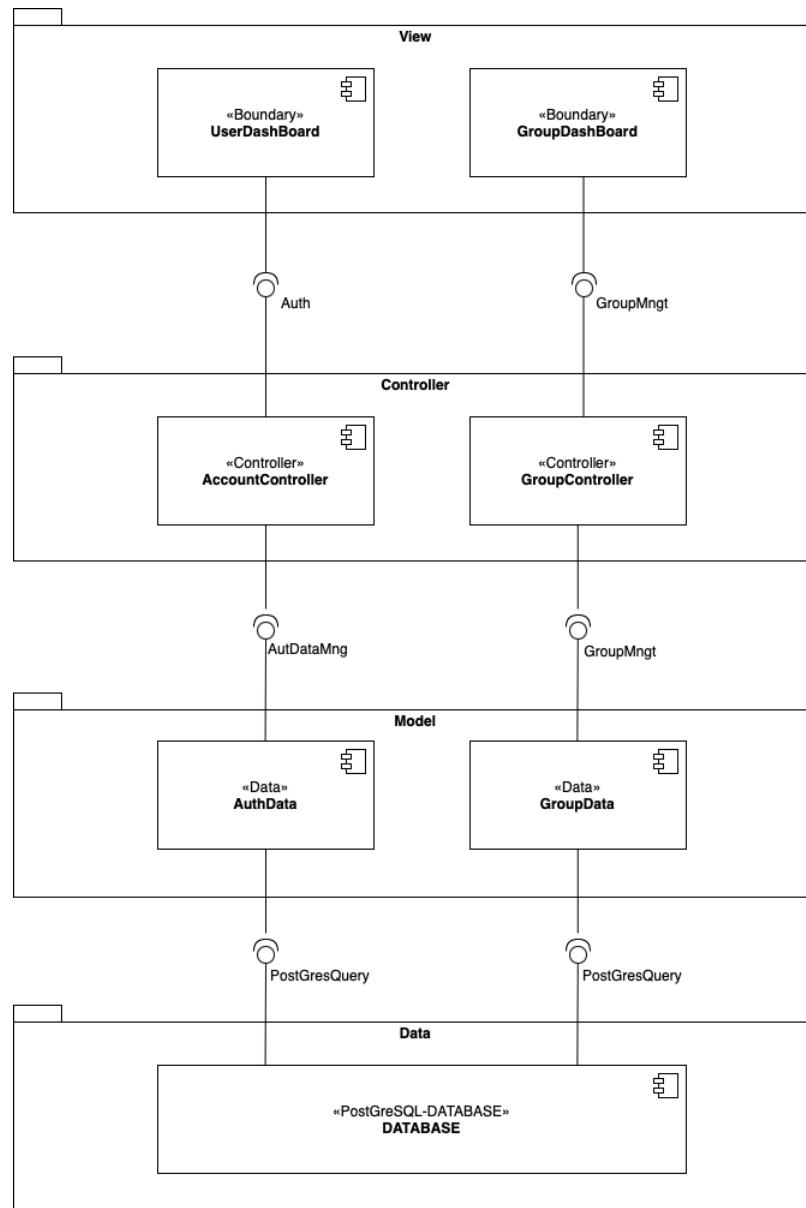
Group → AppUser (admin):

- Rappresenta il ruolo di amministratore per un gruppo.
- Ogni gruppo (**Group**) ha un amministratore specifico (**AppUser**) che è responsabile della gestione del gruppo.
- La relazione è uno-a-uno (1:1), il che significa che ogni gruppo ha un solo amministratore, ma un amministratore può gestire più gruppi.
- Questo legame indica che l'amministratore è una figura centrale per la gestione delle attività e dei membri del gruppo.

AppUser ↔ Group (membri):

- Rappresenta i membri di un gruppo e la loro relazione con i gruppi.
- La relazione è molti-a-molti ($m : n$), il che significa che:
 - Ogni utente (**AppUser**) può essere membro di più gruppi (**Group**).
 - Allo stesso tempo, ogni gruppo può avere più utenti come membri.
- Questo legame mostra che l'utente e il gruppo sono fortemente interconnessi, poiché i gruppi esistono per aggregare utenti, e gli utenti possono partecipare a più attività o comunità rappresentate dai gruppi.

2.4 UML Component Diagram



Partendo dai casi d'uso selezionati per questa iterazione e procedendo con l'utilizzo delle euristiche di design, è stato possibile progettare l'architettura software del sistema **Spendly**. I componenti sono organizzati secondo il pattern **MVC (Model-View-Controller)**, con la suddivisione in:

- **Boundary** - Interfaccia utente, responsabile dell'interazione con l'utente finale.
- **Controller** - Gestione logica di business.
- **Model** - Gestione dei dati e accesso al database.
- **Service** - Servizi di sicurezza e autenticazione.
- **Database** - PostgreSQL.

2.5 UML Deployment Diagram

Mediante l'utilizzo di un Deployment Diagram UML è possibile mostrare la rappresentazione hardware e software del sistema. In Figura 39 vengono mostrati i componenti contenuti nei seguenti nodi:

- **Admin**: Interfaccia web utilizzata da un utente admin per creare, gestire e aggiungere membri ai gruppi. Il nodo contiene il componente *AppUserMngt* per la gestione degli utenti.
- **User**: Interfaccia web utilizzata dagli utenti per accedere alla piattaforma e gestire i gruppi. Contiene i componenti front-end *AppUserMngt* per la gestione dell'account utente e *GroupMngt* per la gestione dei gruppi.
- **Web Server**: Nodo server che espone le API necessarie per la gestione degli utenti e dei gruppi. Contiene i seguenti componenti:
 - *AppUserController*: Controller per la gestione delle operazioni relative agli utenti.
 - *GroupController*: Controller per la gestione delle operazioni sui gruppi.
 - *AppUser*: Modulo dati per la memorizzazione e gestione delle informazioni utente.
 - *Group*: Modulo dati per la memorizzazione e gestione delle informazioni sui gruppi.
- **Database**: PostgreSQL

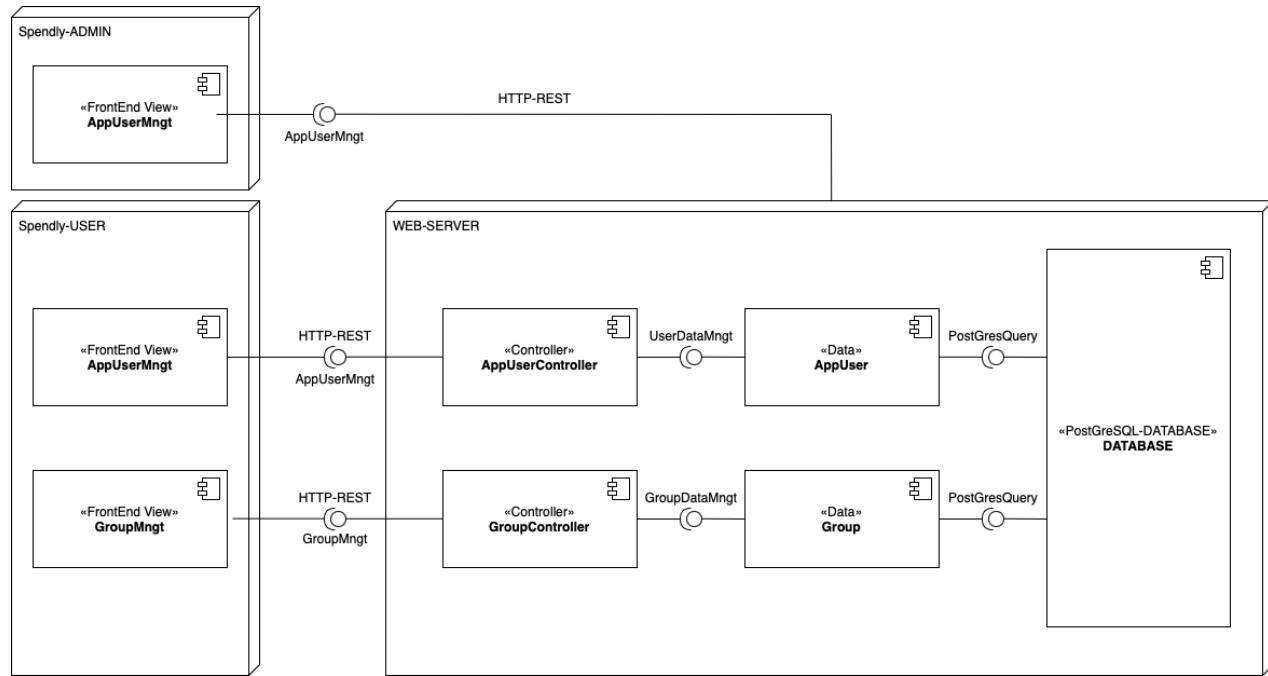


Figura 4: UML Deployment Diagram

2.6 Testing

2.6.1 Analisi Statica - CodeMR

L'analisi statica del codice è stata gestita tramite il tool CodeMR.

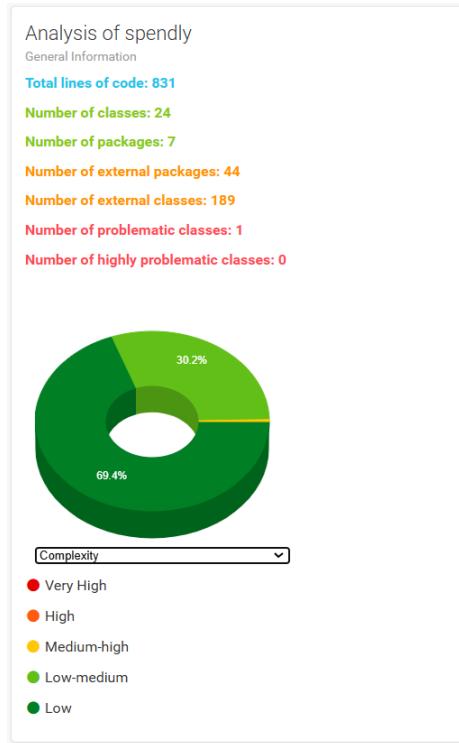


Figura 5: Complexity

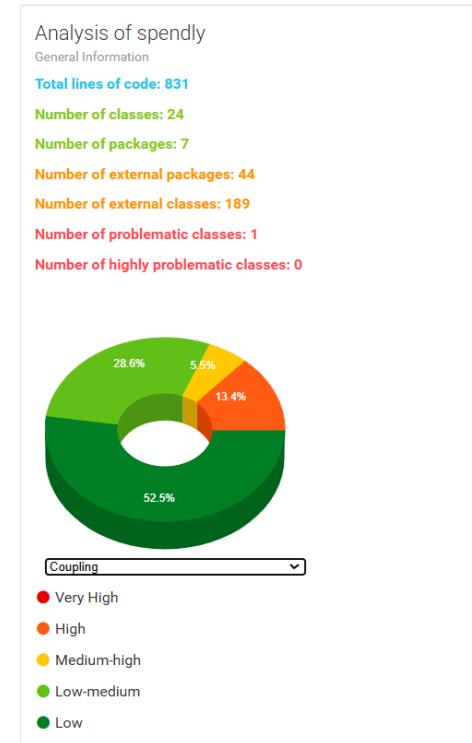


Figura 6: Coupling

Detailed metric tables

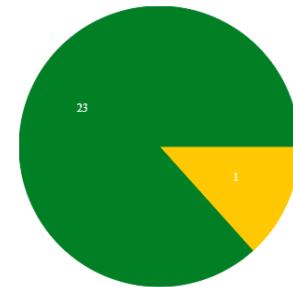
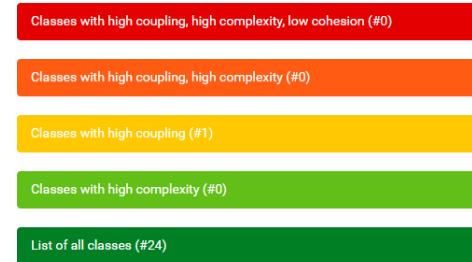


Figura 7: Problemi classi

Classes with high coupling (#1)

ID	CLASS	COUPLING	COMPLEXITY	LACK OF COHESION	SIZE	LOC	CBO APP	CBO LIB	RFC	
1	AccountController					111	21	4	17	59

Figura 8: Problemi classi 2

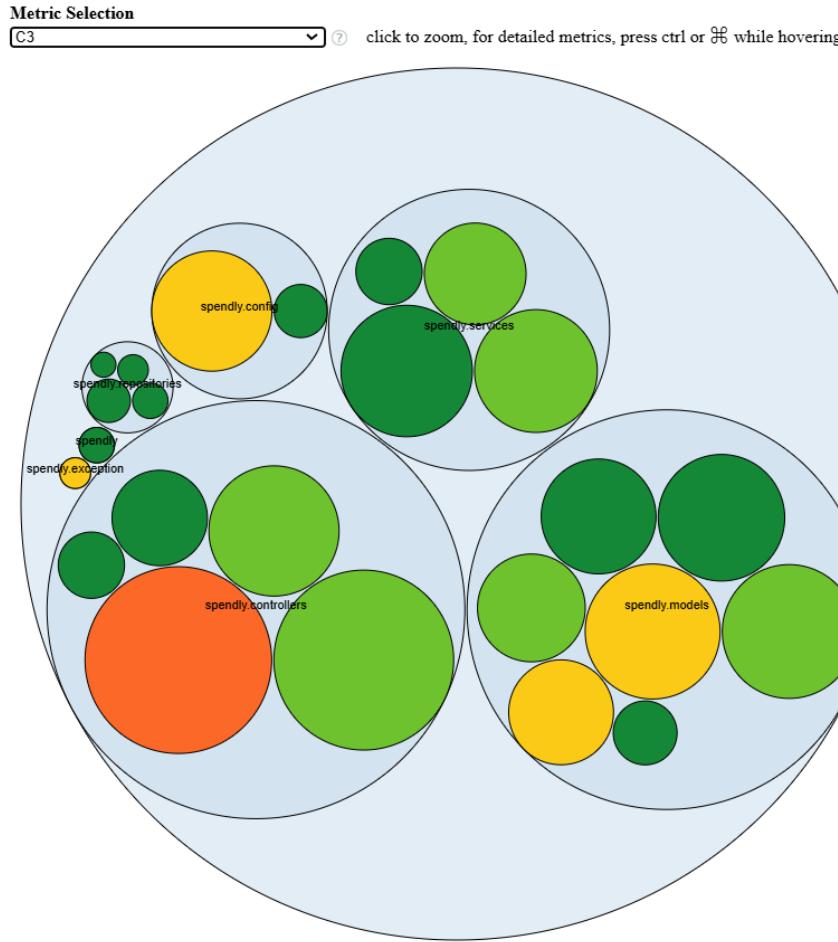


Figura 9: Struttura dei package

2.6.2 Analisi Dinamica - JUnit

L'analisi dinamica del codice è stata condotta utilizzando JUnit per l'esecuzione di test automatizzati sui metodi delle principali classi, e Postman per verificare il corretto funzionamento delle API implementate nei controller. In particolare, con JUnit sono stati sviluppati test specifici per validare i metodi delle classi relative alla gestione dei gruppi, quali Group, GroupService e GroupController.

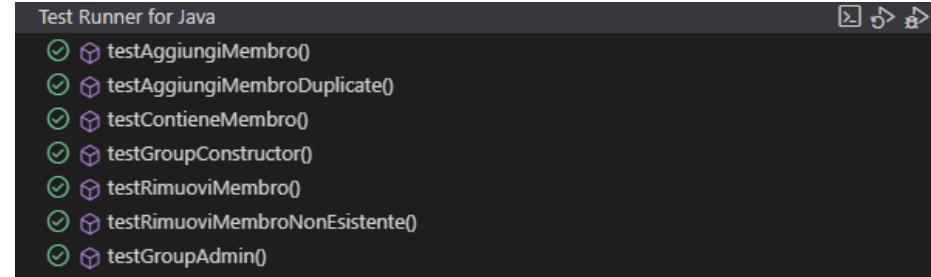


Figura 10: Test per la classe Group

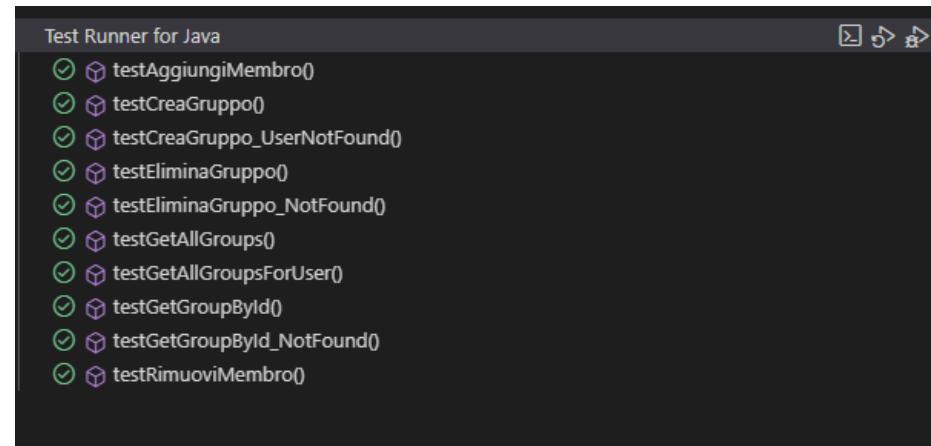


Figura 11: Test per la classe GroupService

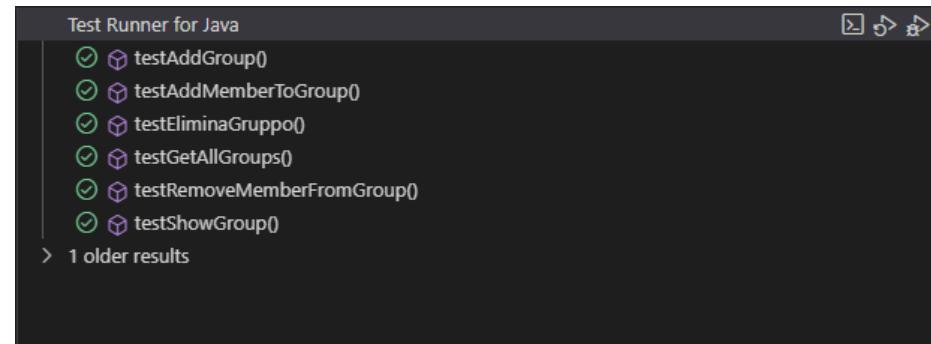


Figura 12: Test per la classe GroupController

2.6.3 API Esposte

Questa sezione documenta le API principali del sistema **Spendly**, includendo autenticazione, gestione dei gruppi e visualizzazione. Ogni test verrà mostrato con un' **immagine dei risultati**.

Registrazione Utente

- **Endpoint:** POST /account/register
- **Descrizione:** Consente a un nuovo utente di registrarsi al sistema.
- **Parametri:**
 - `nome` (string) - Nome utente.
 - `cognome` (string) - Cognome utente.
 - `username` (string) - Username utente (non accetta duplicati).
 - `email` (string) - Email dell'utente (non accetta duplicati).
 - `telefono` (string) - Telefono utente.
 - `indirizzo` (string) - Indirizzo utente.
 - `password` (string) - Password scelta dall'utente.
- **Risultato:**

```

POST      v   http://localhost:8080/account/register

{
    "firstName": "Prova",
    "lastName": "Test",
    "username": "provatest2",
    "email": "prova2@example.com",
    "phone": "+1234567899",
    "address": "123 Main Street, City, Country",
    "password": "securePassword123"
}

{
    "user": {
        "id": 7,
        "firstName": "Prova",
        "lastName": "Test",
        "username": "provatest2",
        "email": "prova2@example.com",
        "phone": null,
        "address": null,
        "password": "$2a$10$8VXPXPFJzK5A403Rrp7CWyuficG336ErgiKpDqXK66pyE5BxAZR/0C",
        "role": "client",
        "createdAt": "2025-02-04T10:17:09.942+00:00"
    },
    "token": "eyJhbGciOiJIUzI1NiJ9.
eyJpc3MiOiJcb29rIFN0b3JlIEFQSSIsInN1YiI6InByb3ZhdGVzd0IiLC3yb2xlijoiY2xpZw50IiwidXhwIjoxNzM4NzUwNjMwLCJpYXQiOjE3Mzg2NjQyMzB9.
p62ufrxdDceze0EdSkHSIU35NrUhUv8reU62cHkrKMR8"
}

```

Figura 13: Risultato API Registrazione

Login Utente

- **Endpoint:** POST /account/login
- **Descrizione:** Permette a un utente registrato di accedere al sistema.
- **Parametri:**
 - **username** (string) - Username dell'utente.
 - **password** (string) - Password dell'utente.
- **Risultato:**

POST http://localhost:8080/account/login

Params Authorization Headers (10) Body Scripts Tests Settings Cookies

Auth Type Bearer Token

The authorization header will be automatically generated when you send the request. Learn more about [Bearer Token](#) authorization.

Token eyJhbGciOiJIUzI1NiJ9eyJpc3MiOiJCb29rF...

```

1 {
2   "username": "provatest2",
3   "password": "securePassword123"
4 }

1 {
2   "user": {
3     "id": 7,
4     "firstName": "Prova",
5     "lastName": "Test",
6     "username": "provatest2",
7     "email": "prova2@example.com",
8     "phone": null,
9     "address": null,
10    "password": "$2a$10$8VPXPfJzK5A403Rrp7CwyuficG336ErgiKpDqXK66pyE5BzAZR/0C",
11    "role": "client",
12    "createdAt": "2025-02-04T10:17:09.942+00:00"
13  },
14  "token": "eyJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJCb29rF...
15 }

```

Figura 14: Risultato API Login

Creazione di un Gruppo

- **Endpoint:** POST /api/groups
- **Descrizione:** Permette la creazione di un nuovo gruppo da parte dell'utente.
- **Parametri:**
 - **name** (string) - Nome del gruppo.
 - **username** (string) - Username dell'utente che crea il gruppo.
- **Risultato:**

HTTP <http://localhost:8080/api/groups?username=MCarr>

POST <http://localhost:8080/api/groups?username=MCarr>

Params • Authorization • Headers (10) Body • Scripts • Settings

none form-data x-www-form-urlencoded raw binary GraphQL **JSON** ▾

```

1  {
2    "nome": "GruppoPostman"
3  }

```

Body Cookies Headers (14) Test Results ⚡

{ } JSON ▾ ▶ Preview 🖥 Visualization

```

1  {
2    "id": 8,
3    "nome": "GruppoPostman",
4    "admin": {
5      "id": 1,
6      "firstName": "Mario",
7      "lastName": "Carraia",
8      "username": "MCarr",
9      "email": "mario.carrara@gmail.com",
10     "phone": null,
11     "address": null,
12     "password": "$2a$10$VamMPosQ5fQmUIjly/TXRujvI9R2bAUISHN.IJJmNsVwPFUnF.MSC",
13     "role": "client",
14     "createdAt": "2025-02-01T15:03:29.402+00:00"
15   }
16 }

```

Figura 15: Risultato API Creazione Gruppo

Inserimento di un utente in un gruppo

- **Endpoint:** POST /api/groups/group_id/members
- **Descrizione:** Permette all'amministratore di inserire un utente in un gruppo.
- **Parametri:**
 - **adminUsername** (string) - Username dell'admin del gruppo.
 - **memberUsername** (string) - Username dell'utente da inserire.
- **Risultato:**

HTTP http://localhost:8080/api/groups/8/members?adminUsername=MCarr&memberUsername=john_doe

POST http://localhost:8080/api/groups/8/members?adminUsername=MCarr&memberUsername=john_doe

Params • Authorization • Headers (9) Body Scripts Settings

Query Params

Key	Value
adminUsername	MCarr
memberUsername	john_doe
Key	Value

Body Cookies Headers (14) Test Results • 

 Raw ▾  Preview  Visualize ▾

1 Member added successfully

Figura 16: Risultato API Inserimento utente

Visualizzazione dei Gruppi

- **Endpoint:** GET /api/groups
- **Descrizione:** Restituisce la lista di tutti i gruppi a cui l'utente appartiene.
- **Parametri:**
 - `username` (string) - Username dell'utente.
- **Risultato:**

HTTP http://localhost:8080/api/groups?username=john_doe

GET http://localhost:8080/api/groups?username=john_doe

Params • Authorization • Headers (8) Body Scripts Settings

Query Params

<input checked="" type="checkbox"/> Key	Value
<input checked="" type="checkbox"/> username	john_doe
Key	Value

Body Cookies Headers (14) Test Results | ⏪

{ } JSON ▾ ▶ Preview ⚡ Visualize ▾

```

1   [
2     {
3       "id": 8,
4       "name": "GruppoPostman",
5       "admin": {
6         "id": 1,
7         "firstName": "Mario",
8         "lastName": "Carrara",
9         "username": "MCarr",
10        "email": "mario.carrara@gmail.com",
11        "phone": null,
12        "address": null,
13        "password": "$2a$10$VamMPOsQ5fQmUIjly/TXRujvI9R2bAUiSHN.IJJmNsVwPFUnF.MSC",
14        "role": "client",
15        "createdAt": "2025-02-01T15:03:29.402+00:00"
      }
    }
  ]

```

Figura 17: Risultato API Visualizzazione Gruppi

3 Iterazione 2

3.1 Casi d'Uso Implementati

In questa iterazione sono stati sviluppati i seguenti casi d'uso ritenuti prioritari per lo sviluppo di **Spendly**.

ID	Titolo
UC4	Crea alert di gruppo
UC5	Modifica alert
UC6	Elimina alert
UC7	Scegli tipologia alert
UC8	Crea gruppo
UC9	Invita membri
UC10	Elimina membri
UC11	Modifica membri
UC12	Elimina gruppo
UC13	Accedi gruppo
UC14	Inserisci spesa
UC15	Elimina spesa
UC16	Modifica spesa
UC17	Scegli tipologia spesa
UC18	Visualizza spese
UC19	Ricalcolo debiti

Tabella 5: Iterazione2 - Casi d'Uso Implementati.

3.1.1 UC4: Creazione Alert di Gruppo

Attori: Utente amministratore, Sistema.

Descrizione: L'amministratore di un gruppo può creare un alert per segnalare il raggiungimento di una soglia limite di spesa.

Flusso degli eventi:

1. L'amministratore accede alla pagina del gruppo.
2. Clicca su "Crea Alert".

3. Inserisce il limite di spesa e una descrizione opzionale.
4. Il sistema salva l'alert e lo associa al gruppo.

3.1.2 UC6: Eliminazione Alert

Attori: Utente amministratore, Sistema.

Descrizione: L'amministratore di un gruppo può eliminare un alert impostato in precedenza.

Flusso degli eventi:

1. L'amministratore accede alla lista degli alert del gruppo.
2. Seleziona un alert e clicca su "Elimina".
3. Il sistema chiede conferma e poi rimuove l'alert.

3.1.3 UC13: Inserimento Spesa

Attori: Utente, Sistema.

Descrizione: Un utente appartenente a un gruppo può aggiungere una spesa condivisa.

Flusso degli eventi:

1. L'utente accede alla pagina del gruppo.
2. Clicca su "Aggiungi Spesa".
3. Inserisce l'importo, la descrizione e seleziona i partecipanti.
4. Il sistema registra la spesa e aggiorna il bilancio del gruppo.

3.1.4 UC14: Eliminazione Spesa

Attori: Utente, Sistema.

Descrizione: Un utente può eliminare una spesa precedentemente inserita.

Flusso degli eventi:

1. L'utente accede alla lista delle spese del gruppo.
2. Seleziona una spesa e clicca su "Elimina".
3. Il sistema chiede conferma e poi rimuove la spesa dal gruppo.

3.1.5 UC15: Modifica Spesa

Attori: Utente, Sistema.

Descrizione: Un utente può modificare i dettagli di una spesa condivisa nel gruppo.

Flusso degli eventi:

1. L'utente accede alla lista delle spese.
2. Seleziona una spesa e clicca su "Modifica".
3. Modifica i dati della spesa (importo, descrizione, partecipanti).
4. Il sistema aggiorna la spesa e ricalcola i bilanci.

3.1.6 UC16: Scegli Tipologia Spesa

Attori: Utente, Sistema.

Descrizione: Un utente può scegliere la tipologia di spesa da inserire nel gruppo.

Flusso degli eventi:

1. L'utente accede alla pagina del gruppo.
2. Clicca su "Scegli Tipologia Spesa".
3. Seleziona la tipologia di spesa desiderata.
4. Il sistema aggiorna la tipologia di spesa selezionata.

3.1.7 UC17: Visualizzazione Spese

Attori: Utente, Sistema.

Descrizione: Un utente può visualizzare tutte le spese del gruppo di cui fa parte.

Flusso degli eventi:

1. L'utente accede alla pagina del gruppo.
2. Clicca su "Visualizza Spese".
3. Il sistema mostra l'elenco delle spese con dettagli su importo, data e partecipanti.

3.1.8 UC18: Ricalcolo Debiti

Attori: Utente, Sistema.

Descrizione: Un utente può calcolare il riepilogo dei debiti tra i membri del gruppo.

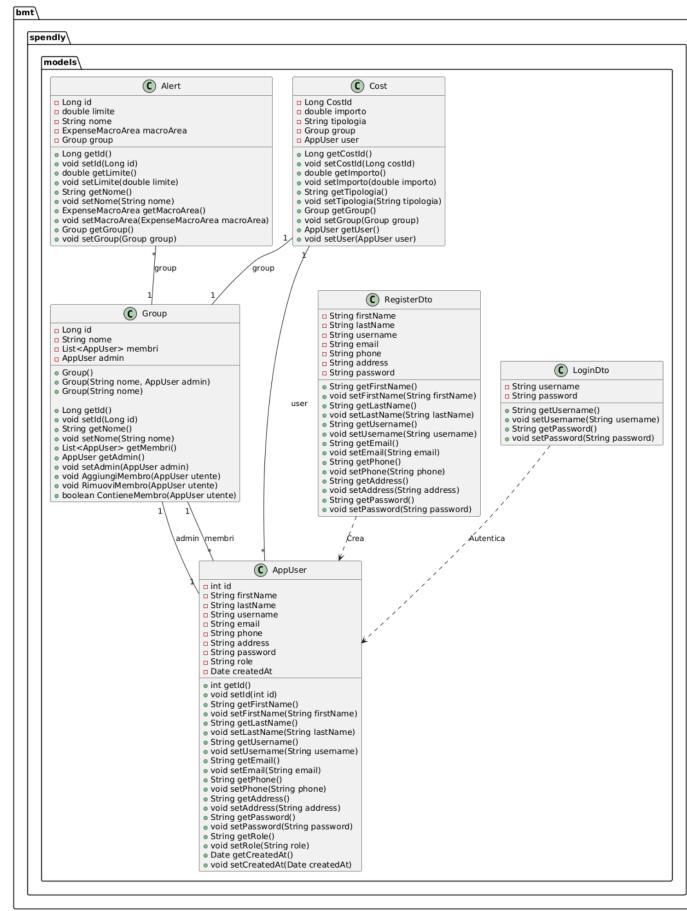
Flusso degli eventi:

1. L'utente accede alla pagina del gruppo.
2. Clicca su "Calcola Debiti".
3. Il sistema analizza le spese e genera un riepilogo dei debiti tra i membri.
4. L'utente visualizza il riepilogo e le modalità di saldo consigliate.

3.2 Diagramma delle Interfacce: Spesa

I CostIF	I CostMngtIF
<ul style="list-style-type: none"> • Long getCostId() • void setCostId(Long costId) • double getImporto() • void setImporto(double importo) • ExpenseType getTipologia() • void setTipologia(ExpenseType tipologia) • Group getGroup() • void setGroup(Group group) • AppUser getUser() • void setUser(AppUser user) • PaymentStatus getPaymentStatus() • void setPaymentStatus(PaymentStatus paymentStatus) 	<ul style="list-style-type: none"> • Cost createCost(Cost cost, Long groupId, String username) • List<Cost> getCostsByUsername(String username) • List<Cost> getCostsByGroup(Long groupId) • List<Cost> getAllCosts() • Cost getCostById(Long id) • Cost updateCost(Long id, Cost updatedCost) • void deleteCost(Long id) • Cost payCostFromBudget(Long costId, String username)
I CostControllerIF <ul style="list-style-type: none"> • ResponseEntity<List<Cost>> getAllCosts(String username) • ResponseEntity<List<Cost>> getCostsByGroup(Long groupId) • ResponseEntity<?> createCost(Cost cost, String username) • ResponseEntity<?> getCostById(Long id) • ResponseEntity<?> updateCost(Long id, Cost updatedCost) • ResponseEntity<?> deleteCost(Long id) • ResponseEntity<?> payCost(Long id, String username) 	

3.3 UML Class Diagram



LoginDto → AppUser:

- Rappresenta il processo di autenticazione nel sistema.
- La relazione mostra che la classe **LoginDto** contiene i dati necessari (username e password) per verificare l'identità di un utente già registrato (**AppUser**).
- In altre parole, **LoginDto** fornisce un mezzo per confermare che un utente (**AppUser**) ha accesso al sistema.

RegisterDto → AppUser:

- Rappresenta il processo di registrazione di un nuovo utente.

- La relazione indica che la classe **RegisterDto** contiene i dati necessari (nome, cognome, email, password, ecc.) per creare un nuovo account utente (**AppUser**) all'interno del sistema.
- Questo significa che la classe **RegisterDto** viene utilizzata per tradurre i dati di registrazione in un'istanza valida della classe **AppUser**.

Group → AppUser (admin):

- Rappresenta il ruolo di amministratore per un gruppo.
- Ogni gruppo (**Group**) ha un amministratore specifico (**AppUser**) che è responsabile della gestione del gruppo.
- La relazione è uno-a-uno (1:1), il che significa che ogni gruppo ha un solo amministratore, ma un amministratore può gestire più gruppi.
- Questo legame indica che l'amministratore è una figura centrale per la gestione delle attività e dei membri del gruppo.

AppUser ↔ Group (membri):

- Rappresenta i membri di un gruppo e la loro relazione con i gruppi.
- La relazione è molti-a-molti ($m : n$), il che significa che:
 - Ogni utente (**AppUser**) può essere membro di più gruppi (**Group**).
 - Allo stesso tempo, ogni gruppo può avere più utenti come membri.
- Questo legame mostra che l'utente e il gruppo sono fortemente interconnessi, poiché i gruppi esistono per aggregare utenti, e gli utenti possono partecipare a più attività o comunità rappresentate dai gruppi.

Cost → Group:

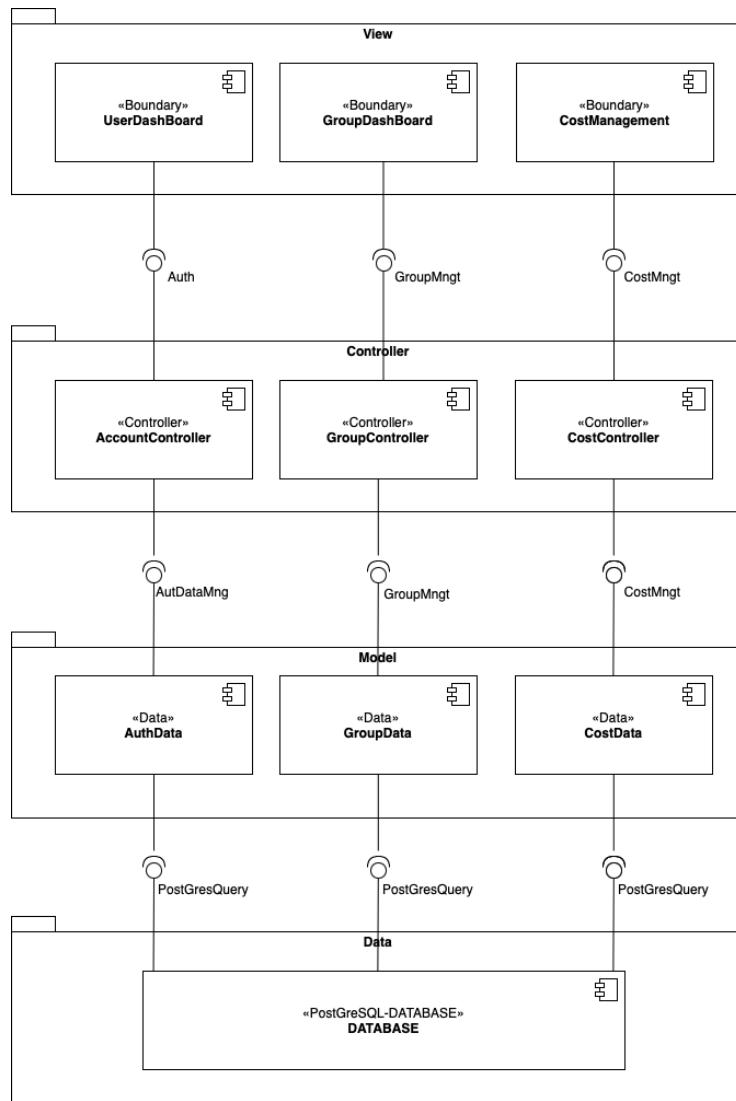
- Indica che un costo (**Cost**) può essere associato a un gruppo (**Group**).
- La relazione è molti-a-uno (1:n), il che significa che:
 - Un costo appartiene a un singolo gruppo.
 - Tuttavia, un gruppo può avere più costi associati.

- Questa relazione consente di rappresentare costi condivisi o transazioni relative a un gruppo specifico.

Cost → AppUser:

- Indica che un costo (**Cost**) è associato a un utente (**AppUser**).
- La relazione è molti-a-uno (1:n), il che significa che:
 - Un costo è creato o sostenuto da un singolo utente.
 - Tuttavia, un utente può avere più costi associati.
- Questa relazione mostra che ogni costo ha un responsabile (l'utente) che lo ha sostenuto.

3.4 UML Component Diagram



In questa iterazione, aggiorniamo il diagramma dei componenti rispetto all'iterazione 1 data l'introduzione dei costi. Partendo dai casi d'uso selezionati per questa iterazione e procedendo con l'utilizzo delle euristiche di design, è stato possibile progettare l'architettura software del sistema **Spendly**.

3.5 UML Deployment Diagram

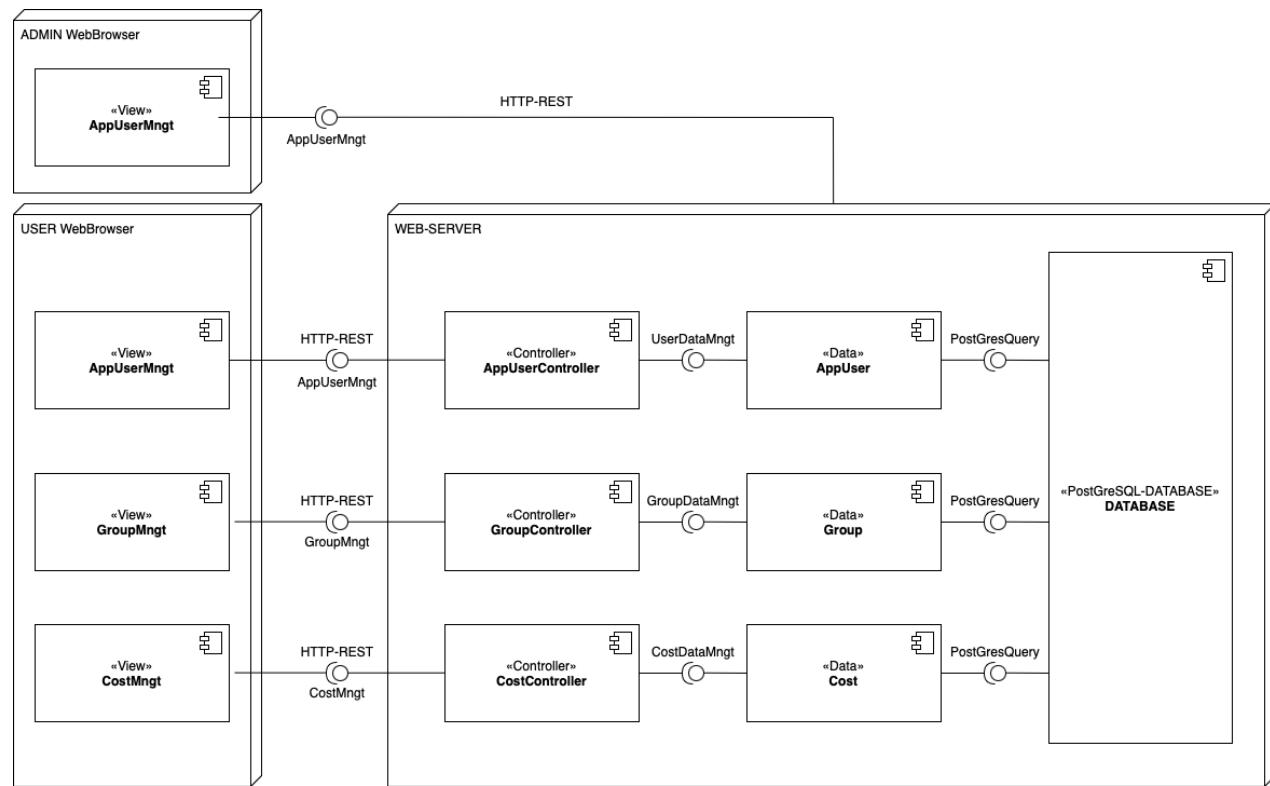


Figura 18: UML Deployment Diagram

3.6 Testing

3.6.1 Analisi Statica - CodeMR

Questa sezione documenta l'analisi statica del codice eseguita con CodeMR per l'iterazione 2.

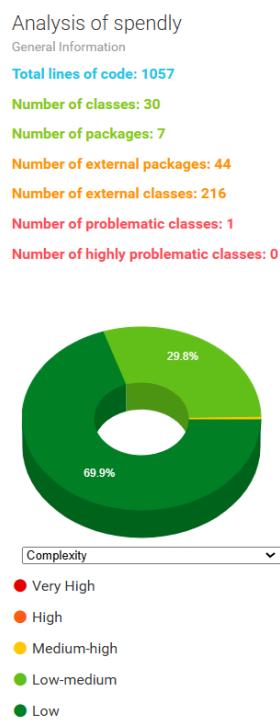


Figura 19: Complexity

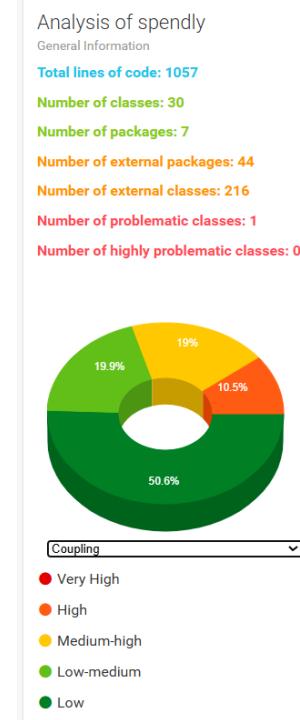


Figura 20: Coupling

Classes with high coupling, high complexity, low cohesion (#0)

ID	CLASS	COUPLING	COMPLEXITY	LACK OF COHESION	SIZE	LOC	WMC	RFC	CBO	LCAM

Classes with high coupling, high complexity (#0)

ID	CLASS	COUPLING	COMPLEXITY	LACK OF COHESION	SIZE	LOC	CBO	WMC	RFC	NOM

Classes with high coupling (#1)

ID	CLASS	COUPLING	COMPLEXITY	LACK OF COHESION	SIZE	LOC	CBO APP	CBO LIB	RFC	
1	AccountController	■	■	■	■	111	21	4	17	59

Classes with high complexity (#0)

ID	CLASS	COUPLING	COMPLEXITY	LACK OF COHESION	SIZE	LOC	WMC	RFC	NOM	DIT

Figura 21: Problemi classi

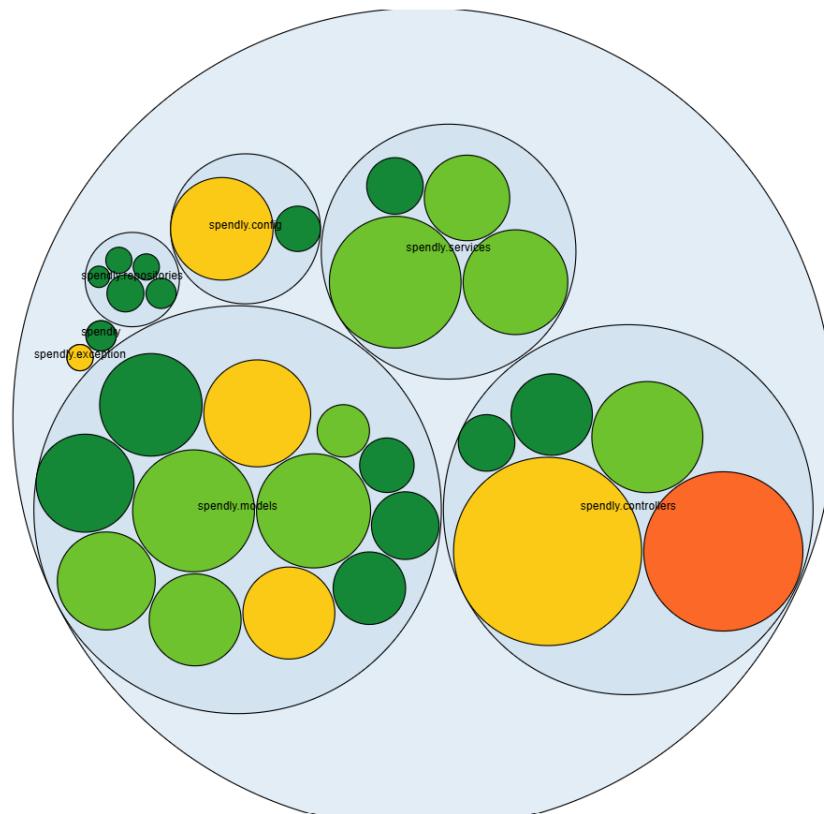


Figura 22: Struttura dei package

3.6.2 Analisi Dinamica - JUnit

Questa sezione documenta i test d'unità implementati per l'iterazione 2. Sono stati aggiornati i test riguardanti i gruppi e sono stati implementati i test relativi ai costi.

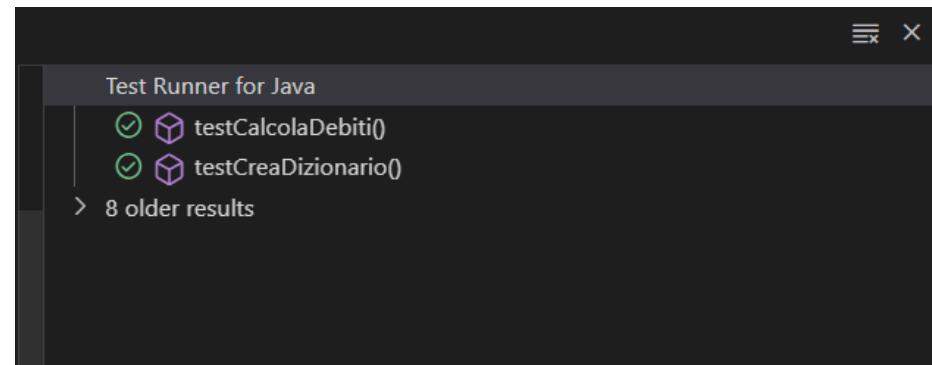


Figura 23: Test per ottimizzare Debiti

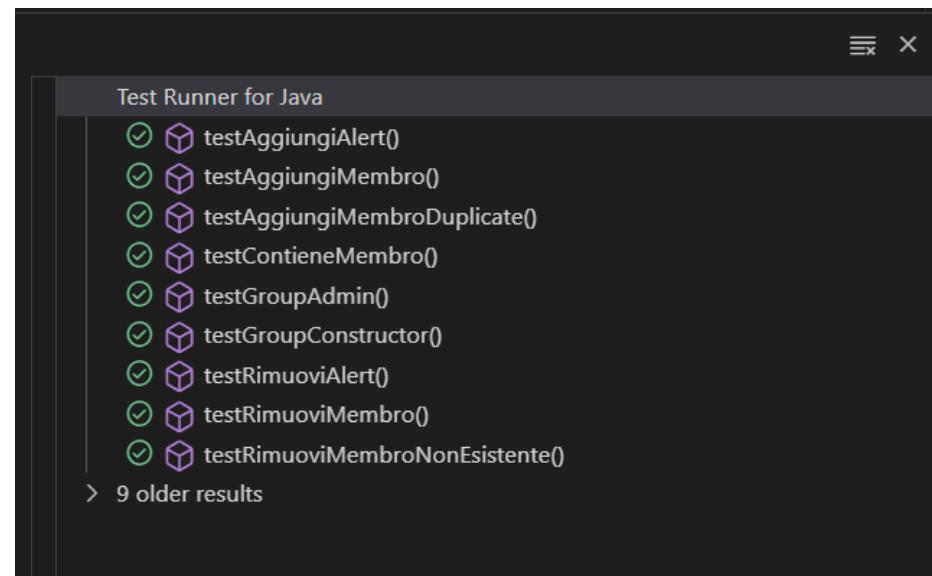


Figura 24: Test aggiornati per la classe Group

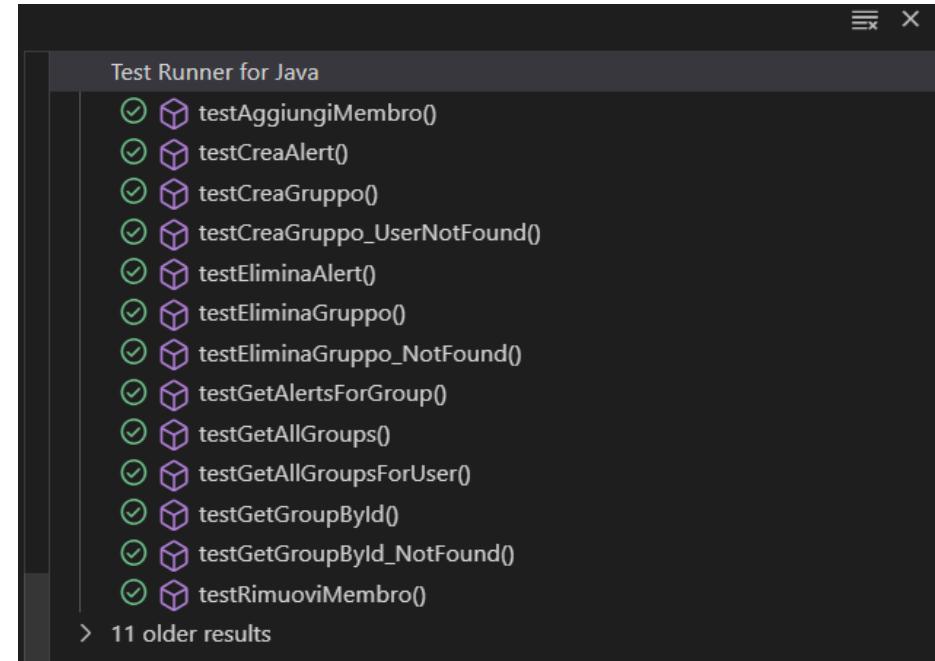


Figura 25: Test aggiornati per la classe GroupService

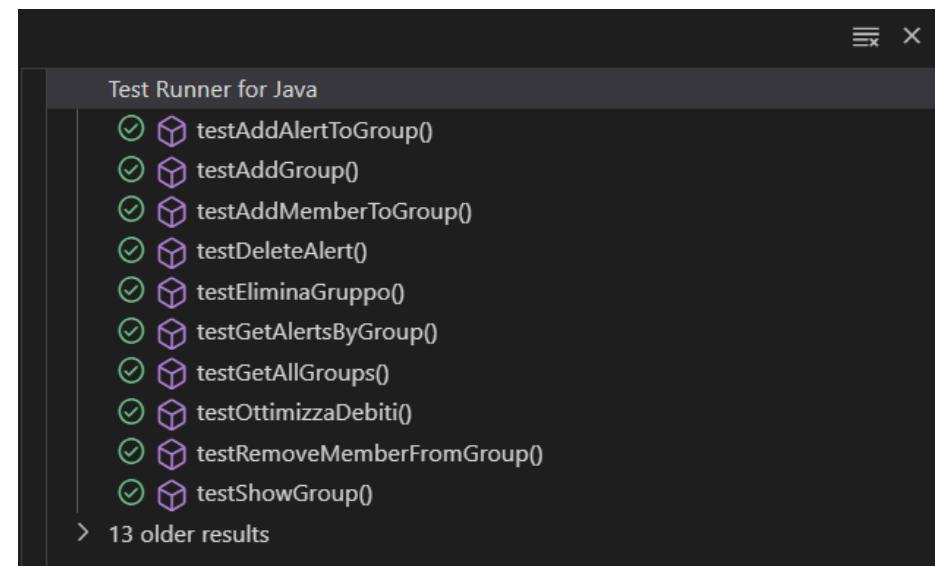


Figura 26: Test aggiornati per la classe GroupController

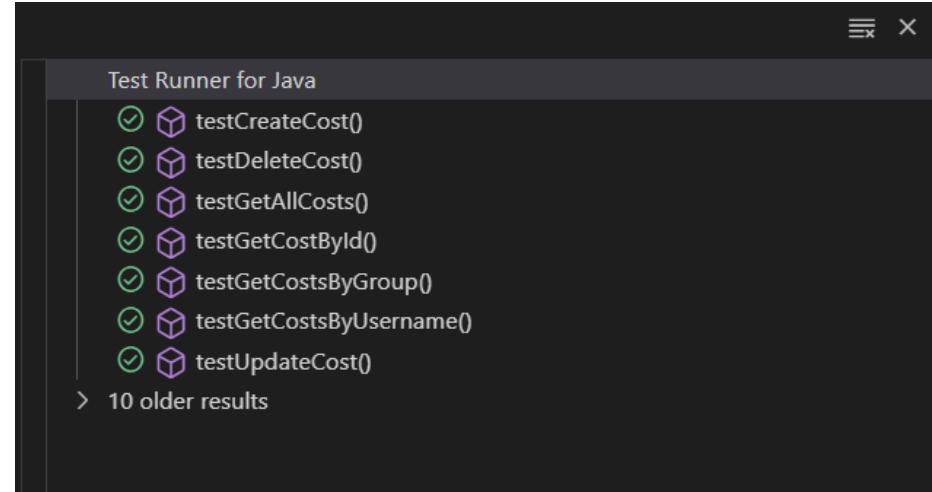


Figura 27: Test per la classe CostService

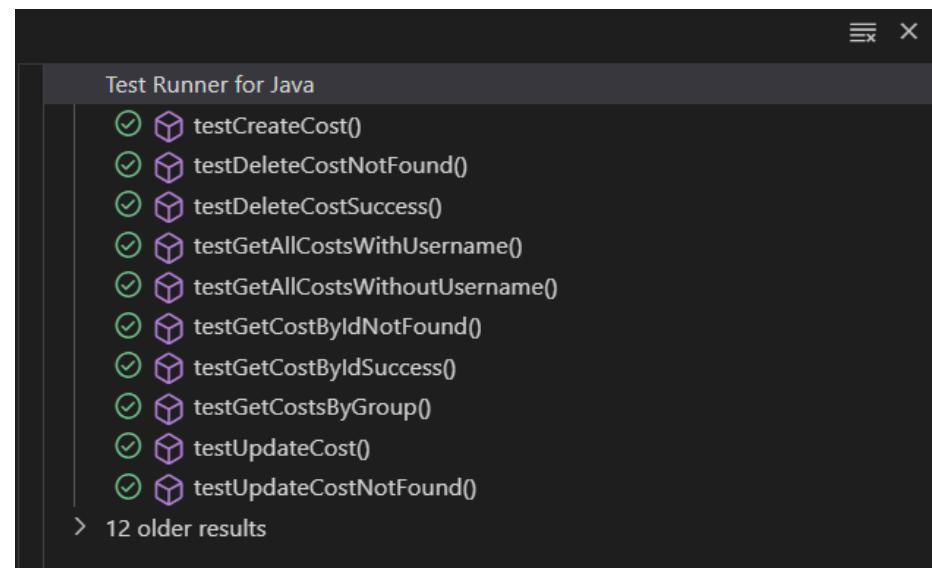


Figura 28: Test per la classe CostController

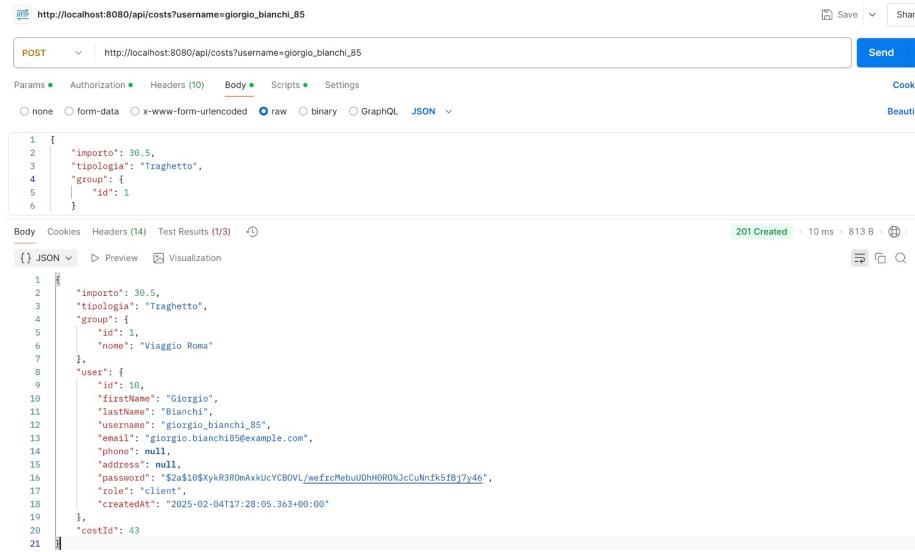
3.6.3 API Costi

Questa sezione documenta le API relative alla gestione dei costi e degli alert nel sistema Spendly, inclusi la creazione, l'eliminazione e la visualizzazione

dei costi di utenti e gruppi. Ogni test verrà mostrato con un'immagine dei risultati.

Aggiunta di un Costo

- **Endpoint:** POST /api/costs?username={username}
- **Descrizione:** Permette all'utente autenticato di aggiungere un nuovo costo, eventualmente associandolo a un gruppo.
- **Parametri:**
 - **importo** (double) - Importo della spesa.
 - **tipologia** (string) - Tipo di spesa.
 - **groupId** (integer, opzionale) - ID del gruppo a cui associare il costo.



```

1 {
2   "importo": 30.5,
3   "tipologia": "Traghetto",
4   "group": {
5     "id": 1
6   }
}
    
```

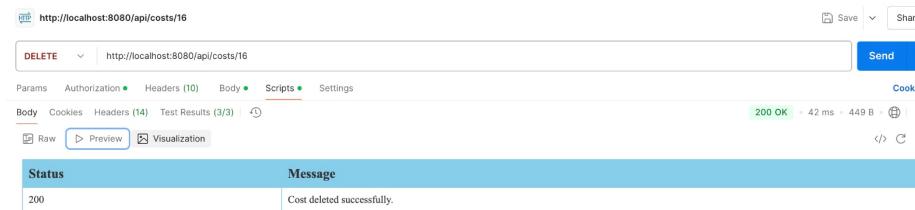
Body Cookies Headers (14) Test Results (1/3)

201 Created · 10 ms · 813 B

Figura 29: Risultato API Aggiunta Costo

Eliminazione di un Costo

- **Endpoint:** DELETE /api/costs/{costId}
- **Descrizione:** Permette all’utente autenticato di eliminare un costo precedentemente registrato.
- **Parametri:**
 - {costId} (integer) - ID del costo da eliminare.

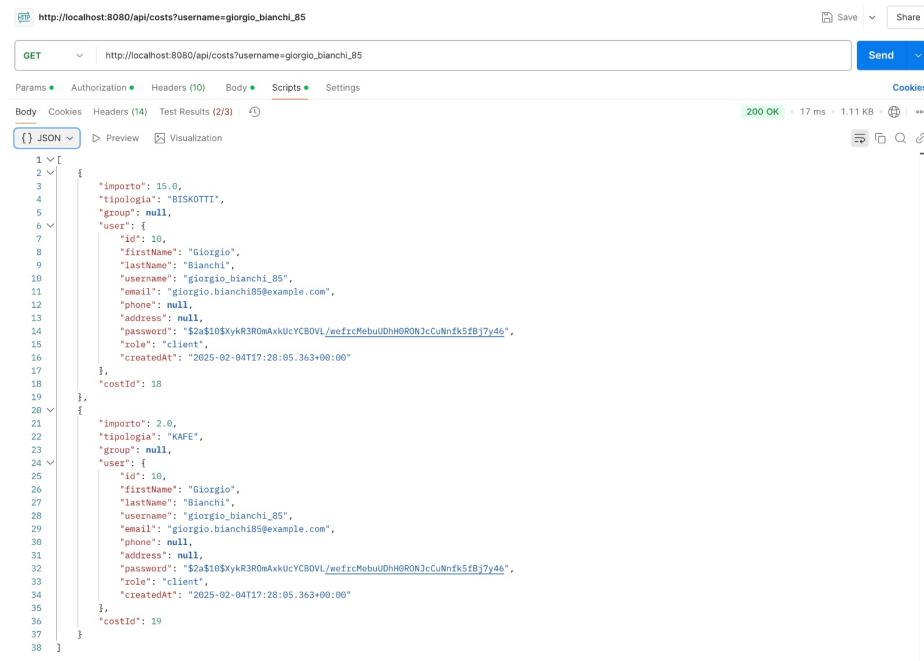


Status	Message
200	Cost deleted successfully.

Figura 30: Risultato API Eliminazione Costo

Visualizzazione Costi Utente

- **Endpoint:** GET /api/costs?username={username}
- **Descrizione:** Restituisce la lista di tutti i costi registrati dall'utente autenticato.



The screenshot shows a Postman request for the URL `http://localhost:8080/api/costs?username=giorgio_bianchi_85`. The response status is `200 OK` with a response time of 17 ms and a size of 1.11 KB. The response body is displayed in JSON format, showing two cost objects. Each cost object has properties like `importo`, `tipologia`, `groupp`, and a nested `user` object containing details such as `firstName`, `lastName`, `username`, `email`, `password`, `role`, and `createdAt`.

```

1  [
2   {
3     "importo": 15.0,
4     "tipologia": "BITSKOTTI",
5     "groupp": null,
6     "user": {
7       "id": 10,
8       "firstName": "Giorgio",
9       "lastName": "Bianchi",
10      "username": "giorgio_bianchi_85",
11      "email": "giorgio.bianchi85@example.com",
12      "password": "$2a$10$XykR3R0nAvkUcYCB0VL/wefrcMebuUDHORONJcCuNnfk5fBj7y46",
13      "role": "client",
14      "createdAt": "2025-02-04T17:28:05.363+00:00"
15    },
16    "costId": 18
17  },
18  {
19    "importo": 2.0,
20    "tipologia": "KAFE",
21    "groupp": null,
22    "user": {
23      "id": 10,
24      "firstName": "Giorgio",
25      "lastName": "Bianchi",
26      "username": "giorgio_bianchi_85",
27      "email": "giorgio.bianchi85@example.com",
28      "password": "$2a$10$XykR3R0nAvkUcYCB0VL/wefrcMebuUDHORONJcCuNnfk5fBj7y46",
29      "role": "client",
30      "createdAt": "2025-02-04T17:28:05.363+00:00"
31    },
32    "costId": 19
33  }
34 ]

```

Figura 31: Risultato API Visualizzazione Costi Utente

Visualizzazione Costi di un Gruppo

- **Endpoint:** GET /api/costs/group/{groupId}
- **Descrizione:** Restituisce la lista di tutti i costi registrati del gruppo.

http://localhost:8080/api/costs/group/1

Save Share

GET http://localhost:8080/api/costs/group/1

Params Authorization Headers (6) Body Scripts Settings Cookies

Body Cookies Headers (14) Test Results ⚙️

{ } JSON D Preview 📐 Visualization

200 OK 21 ms 1.92 KB ⚙️

↪ C ⌂

Importo	Tipologia	Group ID	Group Name	User ID	First Name	Last Name	Username	Email	Phone	Address	Role	Created At	Cost ID
30.5	trasporto	1	Viaggio Roma	10	Giorgio	Bianchi	giorgio_bianchi_85	giorgio.bianchi85@example.com			client	2025-02-04T17:28:05.363+00:00	38
30.5	AVIONN	1	Viaggio Roma	10	Giorgio	Bianchi	giorgio_bianchi_85	giorgio.bianchi85@example.com			client	2025-02-04T17:28:05.363+00:00	41
30.5	AVIONN	1	Viaggio Roma	10	Giorgio	Bianchi	giorgio_bianchi_85	giorgio.bianchi85@example.com			client	2025-02-04T17:28:05.363+00:00	42
30.5	Traghetto	1	Viaggio Roma	10	Giorgio	Bianchi	giorgio_bianchi_85	giorgio.bianchi85@example.com			client	2025-02-04T17:28:05.363+00:00	43

Figura 32: Risultato API Visualizzazione Costi Gruppo

Creazione Alert

- **Endpoint:** POST /api/groups/{groupId}/alerts
- **Descrizione:** Permette all'amministratore del gruppo di creare un Alert.
- **Parametri:**
 - **adminUsername** (String) - Username admin del gruppo.
 - **nome** (String) -Nome alert.
 - **limite** (double) - Limite alert.
 - **macroArea** (enum) - Area alert.

POST <http://localhost:8080/api/groups/8/alerts?adminUsername=MCarr>

Params • Authorization • Headers (9) **Body** • Scripts Settings

none form-data x-www-form-urlencoded raw binary GraphQL **JSON** ▾

```

1  {
2    "nome": "Limite Spese Casa",
3    "limite": 500,
4    "macroArea": "ABITAZIONE"
5  }

```

Body Cookies Headers (14) Test Results | ⏱

{ } JSON ▾ ▷ Preview ⚡ Visualize | ▾

```

1  {
2    "id": 9,
3    "limite": 500.0,
4    "nome": "Limite Spese Casa",
5    "macroArea": "ABITAZIONE",
6    "group": {
7      "id": 8,
8      "nome": "GruppoPostman",
9      "membri": [

```

Figura 33: Risultato API Creazione Alert

Eliminazione Alert

- **Endpoint:** DELETE /api/groups/groupId/alerts/alertId
- **Descrizione:** Permette all'amministratore del gruppo di eliminare un Alert.
- **Parametri:**
 - adminUsername (String) - Username admin del gruppo.

HTTP <http://localhost:8080/api/groups/8/alerts/2?adminUsername=MCarr>

DELETE <http://localhost:8080/api/groups/8/alerts/2?adminUsername=MCarr>

Params • Authorization • Headers (9) Body • Scripts Settings

Query Params

	Key	Value
<input checked="" type="checkbox"/>	adminUsername	MCarr
	Key	Value

Body Cookies Headers (14) Test Results | ⌂

Raw ▾ Preview ⌂ Visualize | ▾

```
1 Alert deleted successfully
```

Figura 34: Risultato API eliminazione Alert

Visualizza Alert di Gruppo

- **Endpoint:** GET /api/groups/groupId/alerts
- **Descrizione:** Restituisce la lista di tutti gli alerts del gruppo.

HTTP <http://localhost:8080/api/groups/8/alerts>

GET <http://localhost:8080/api/groups/8/alerts>

Params Authorization (1) Headers (9) Body (1) Scripts Settings

Query Params

Key	Value
Key	Value

Body Cookies Headers (14) Test Results

{ } JSON ▾ ▶ Preview ⚡ Visualize

```
1 [  
2 {  
3     "id": 2,  
4     "limite": 500.0,  
5     "nome": "Spesa Mensile",  
6     "macroArea": null,  
7     "group": {  
8         "id": 8,  
9         "nome": "GruppoPostman",  
10        "membri": [  
11            {  
12                "id": 1,  
13                "nome": "Postman",  
14                "email": "postman@postman.com",  
15                "ruolo": "Amministratore",  
16                "gruppi": [  
17                    {  
18                        "id": 1,  
19                        "nome": "Spesa Mensile",  
20                        "macroArea": null,  
21                        "group": null  
22                    }  
23                ]  
24            }  
25        ]  
26    }  
27 }
```

Figura 35: Risultato API Visualizzazione Alert di Gruppo

3.7 Algoritmo ottimizza debiti

3.7.1 PseudoCodice

```
creaDizionario(ListaSpese):
    Dizionario credit <- Dizionario (utente, credit_score)
    inizializzo tutti i credi score a 0

    for item in ListaSpese:
        pagante = item.pagante
        beneficiari[] = item.beneficiari
        importo = item.importo
        quota = importo / len(beneficiari)

        // Se il pagante è tra i beneficiari, deve recuperare solo la quota degli altri
        if pagante in beneficiari:
            credit_score[pagante] += importo - quota
            beneficiari.remove(pagante) // Evito di sottrargli la quota due volte
        else:
            credit_score[pagante] += importo

        // Aggiorno i crediti dei beneficiari sottraendo la loro quota
        for beneficiario in beneficiari:
            credit_score[beneficiario] -= quota

    return credit
```

Figura 36: Funzione CreaDizionario

```

calcolaDebiti(ListaSpese):
    Dizionario credit_score = creaDizionario(ListaSpese)
    Dizionario positivi = {} // Lista di chi deve ricevere soldi
    Dizionario negativi = {} // Lista di chi deve dare soldi

    // Separiamo i creditori dai debitori
    for each utente in credit_score:
        if(credit_score[utente]>0) positivi.add(utente,credit_score[utente])
        if(credit_score[utente]<0) negativi.add(utente,credit_score[utente])

    Ordino i negativi in ordine crescente //( es: -50, -30, 20, -5)
    lista_transazioni = []

    // Associo creditori e debitori
    for positivo in positivi:
        credito=credit_score[positivo]
        i = 0
        while i < len(negativi) and credito > 0:
            negativo = negativi[i]
            debito_assoluto = abs(negativo.credit_score)

            if debito_assoluto <= credito:
                lista_transazioni.add((negativo.utente, positivo.utente, debito_assoluto))
                credito -= debito_assoluto
                negativi.remove(negativo) // Rimuovo il debitore se il debito è saldato
            else:
                lista_transazioni.add((negativo.utente, positivo.utente, credito))
                negativi[i].credit_score += credito
                credito = 0

            if credito == 0:
                break // Passo al prossimo creditore

        i += 1 // Avanzo al prossimo debitore solo se quello attuale è stato saldato

    return lista_transazioni

```

Figura 37: Funzione CalcolaDebiti

3.7.2 Descrizione

La funzione `CreaDizionario` prende in input una matrice di spese e costruisce un dizionario in cui:

- La chiave è lo *username* dell'utente.
- Il valore è il suo *credit score*, che indica quanto deve ricevere o restituire.

Scorrendo i record della matrice, la funzione aggiorna progressivamente i *credit score* degli utenti.

Se il *credit score* è positivo, l'utente deve ricevere soldi. Se invece è negativo, deve restituire una somma.

La funzione `calcolaDebiti` utilizza `CreaDizionario` per ottenere il saldo di ogni utente. A questo punto, suddivide gli utenti in due liste:

- **Creditori:** utenti con *credit score* positivo.
- **Debitori:** utenti con *credit score* negativo, ordinati in modo crescente.

Per ogni creditore, si scorre l'elenco dei debitori per saldare i crediti:

1. Se il debito di un utente è minore o uguale al credito disponibile:
 - Si sottrae l'intero importo dal credito.
 - La transazione viene registrata.
 - Il debitore viene rimosso dalla lista, poiché ha estinto il suo debito.
 - Se il credito arriva a zero, si interrompe l'iterazione e si passa al creditore successivo.
2. Se invece il debito è maggiore del credito:
 - Si utilizza tutto il credito disponibile per ridurre il debito.
 - La transazione viene registrata.
 - Il debitore rimane nella lista, poiché ha ancora un saldo negativo da coprire.
 - Il credito diventa zero e si passa al creditore successivo.

3.7.3 Complessità

La funzione `CreaDizionario` ha:

- Un ciclo esterno che itera su tutti gli elementi della matrice → $O(n)$
- Un ciclo interno che itera su tutti i beneficiari di ogni record → $O(n)$

Essendo i due cicli annidati, la complessità totale è: $O(n^2)$.

La funzione `CalcolaDebiti` esegue le seguenti operazioni:

1. Richiama **CreaDizionario** $\rightarrow O(n^2)$
2. Itera sul dizionario per separare creditori e debitori $\rightarrow O(n)$
3. Ordina la lista dei debitori $\rightarrow O(n \log n)$
4. Associa creditori e debitori
 - Itera su tutti i creditori $\rightarrow O(n)$
 - Per ogni creditore, nel caso peggiore, può iterare su tutti i debitori $\rightarrow O(n)$
 - Complessità totale del ciclo annidato $\rightarrow O(n^2)$

Sommiamo i contributi principali:

$$O(n^2) + O(n) + O(n \log n) + O(n^2) = O(n^2)$$

Complessità finale: $O(n^2)$

4 Iterazione 3

4.1 Introduzione

In questa iterazione, abbiamo implementato la gestione del Budget, permettendo agli utenti di monitorare e gestire i propri fondi disponibili. Inoltre, abbiamo introdotto la funzionalità di Risparmio, che consente agli utenti di creare obiettivi di risparmio per mettere da parte denaro per esigenze future.

4.2 Casi d'Uso

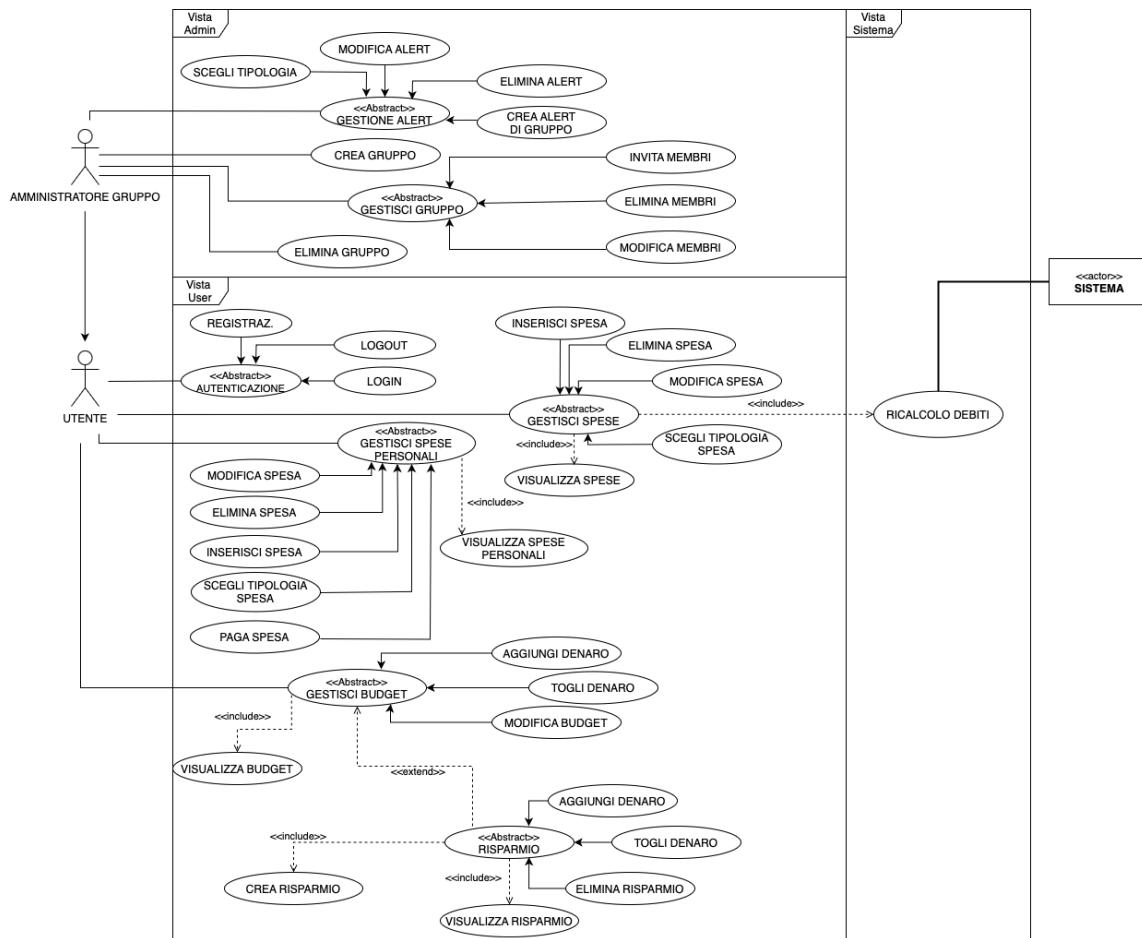


Figura 38: Diagramma casi d'uso

4.2.1 Casi d'uso Implementati

Nota: In Iterazione3, si è mantenuto il numero dei casi d'uso inerenti alle spese personali e al budget (da UC20 a UC28 come in CasiDuso.tex) e sono stati aggiunti i casi di gestione del risparmio, numerati da UC29 a UC32, in quanto non presenti in Iterazione0.

ID	Titolo
UC20	Inserisci spesa personale
UC21	Elimina spesa personale
UC22	Modifica spesa personale
UC23	Scegli tipologia spesa personale
UC24	Paga spesa personale
UC25	Visualizza spesa personale
UC26	Aggiungi denaro a budget
UC27	Togli denaro da budget
UC28	Visualizza budget
UC29	Crea risparmio
UC30	Visualizza risparmio
UC31	Modifica risparmio
UC32	Elimina risparmio

Tabella 6: Iterazione 3 - Casi d'Uso Implementati.

4.2.2 UC20: Inserimento Spesa Personale

Attori: Utente, Sistema.

Descrizione: Un utente può aggiungere una spesa personale.

Flusso degli eventi:

1. L'utente accede alla sezione delle spese personali.
2. Clicca su “Aggiungi Spesa”.
3. Inserisce l'importo, la descrizione e la data.
4. Il sistema registra la spesa e aggiorna il budget.

4.2.3 UC21: Eliminazione Spesa Personale

Attori: Utente, Sistema.

Descrizione: Un utente può eliminare una spesa personale.

Flusso degli eventi:

1. L'utente accede alla lista delle spese personali.
2. Seleziona una spesa e clicca su “Elimina”.
3. Il sistema chiede conferma e rimuove la spesa.

4.2.4 UC22: Modifica Spesa Personale

Attori: Utente, Sistema.

Descrizione: Un utente può modificare i dettagli di una spesa personale.

Flusso degli eventi:

1. L'utente accede alla lista delle spese personali.
2. Seleziona una spesa e clicca su “Modifica”.
3. Modifica i dati (importo, descrizione, data).
4. Il sistema aggiorna la spesa.

4.2.5 UC23: Scegli Tipologia Spesa Personale

Attori: Utente, Sistema.

Descrizione: Un utente può scegliere la tipologia di una spesa personale.

Flusso degli eventi:

1. L'utente accede alla sezione delle spese personali.
2. Clicca su “Scegli Tipologia”.
3. Seleziona la tipologia della spesa.
4. Il sistema aggiorna la tipologia della spesa.

4.2.6 UC24: Pagamento Spesa Personale

Attori: Utente, Sistema.

Descrizione: Un utente può saldare una spesa personale direttamente tramite i metodi di pagamento disponibili.

Flusso degli eventi:

1. L'utente accede alla lista delle spese personali.
2. Seleziona una spesa e clicca su "Paga".
3. Il sistema elabora il pagamento e aggiorna il budget scalando l'importo.

4.2.7 UC25: Visualizzazione Spesa Personale

Attori: Utente, Sistema.

Descrizione: Un utente può visualizzare tutte le proprie spese personali.

Flusso degli eventi:

1. L'utente accede alla sezione delle spese personali.
2. Clicca su "Visualizza Spese".
3. Il sistema mostra l'elenco delle spese con dettagli.

4.2.8 UC26: Aggiunta Denaro a Budget

Attori: Utente, Sistema.

Descrizione: Un utente può aggiungere denaro al budget disponibile.

Flusso degli eventi:

1. L'utente accede alla sezione budget.
2. Clicca su "Aggiungi Denaro".
3. Inserisce l'importo.
4. Il sistema aggiorna il budget disponibile.

4.2.9 UC27: Togli Denaro da Budget

Attori: Utente, Sistema.

Descrizione: Un utente può togliere denaro dal budget disponibile.

Flusso degli eventi:

1. L'utente accede alla sezione budget.
2. Clicca su “Togli Denaro”.
3. Inserisce l'importo.
4. Il sistema aggiorna il budget disponibile.

4.2.10 UC28: Visualizzazione Budget

Attori: Utente, Sistema.

Descrizione: Un utente può visualizzare il budget disponibile e le spese totali.

Flusso degli eventi:

1. L'utente accede alla sezione budget.
2. Il sistema mostra l'importo disponibile e le spese sostenute.

4.2.11 UC29: Creazione Risparmio

Attori: Utente, Sistema.

Descrizione: Un utente può creare un piano di risparmio con un obiettivo definito.

Flusso degli eventi:

1. L'utente accede alla sezione risparmi.
2. Clicca su “Crea Risparmio”.
3. Inserisce l'obiettivo e l'importo.
4. Il sistema registra il piano e aggiorna i dati.

4.2.12 UC30: Visualizzazione Risparmio

Attori: Utente, Sistema.

Descrizione: Un utente può visualizzare i piani di risparmio esistenti.

Flusso degli eventi:

1. L'utente accede alla sezione risparmi.
2. Il sistema mostra l'elenco dei risparmi con dettagli.

4.2.13 UC31: Modifica Risparmio

Attori: Utente, Sistema.

Descrizione: Un utente può modificare i dettagli di un piano di risparmio.

Flusso degli eventi:

1. L'utente accede alla lista dei risparmi.
2. Seleziona un piano e clicca su “Modifica” .
3. Modifica l'importo o la descrizione.
4. Il sistema aggiorna le informazioni.

4.2.14 UC32: Elimina Risparmio

Attori: Utente, Sistema.

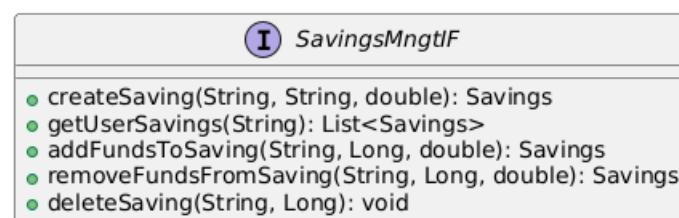
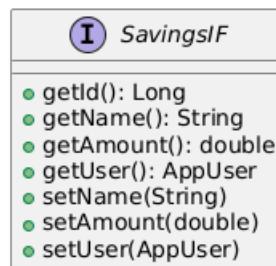
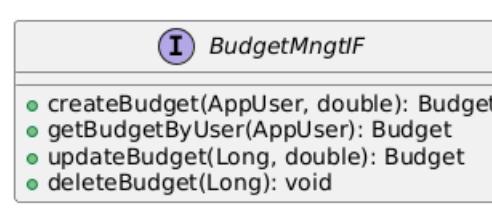
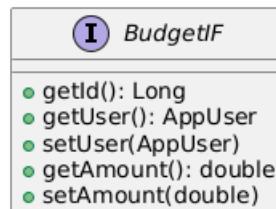
Descrizione: Un utente può eliminare un piano di risparmio, restituendo l'importo relativo al budget.

Flusso degli eventi:

1. L'utente accede alla lista dei risparmi.
2. Seleziona un risparmio e clicca su “Elimina” .
3. Il sistema chiede conferma e, se confermato, rimuove il risparmio re-staurando l'importo al budget.

4.3 Diagramma delle Interfacce

Mostriamo di seguito il diagramma delle interfacce di Budget e Risparmio, implementate in questa iterazione.

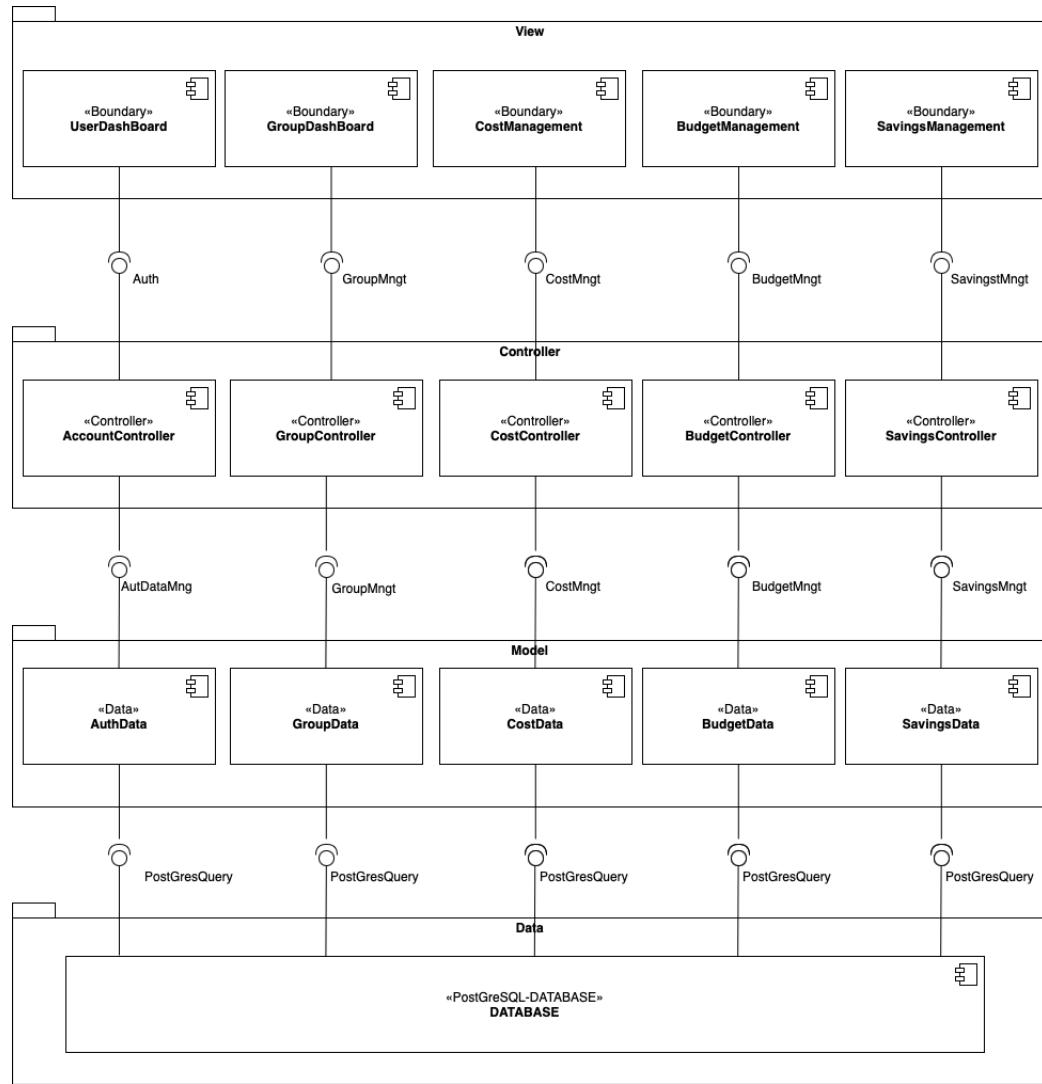


4.4 UML Class Diagram per tipi di dato

Budget
Long : id
AppUser: user
double: amount

Risparmio
Long : id
String: name
double: amount
AppUser: user

4.5 UML Component Diagram



Aggiorniamo anche l'UML Component Diagram aggiungendo i componenti relativi alla gestione del Budget e del Risparmio.

4.6 UML Deployment Diagram

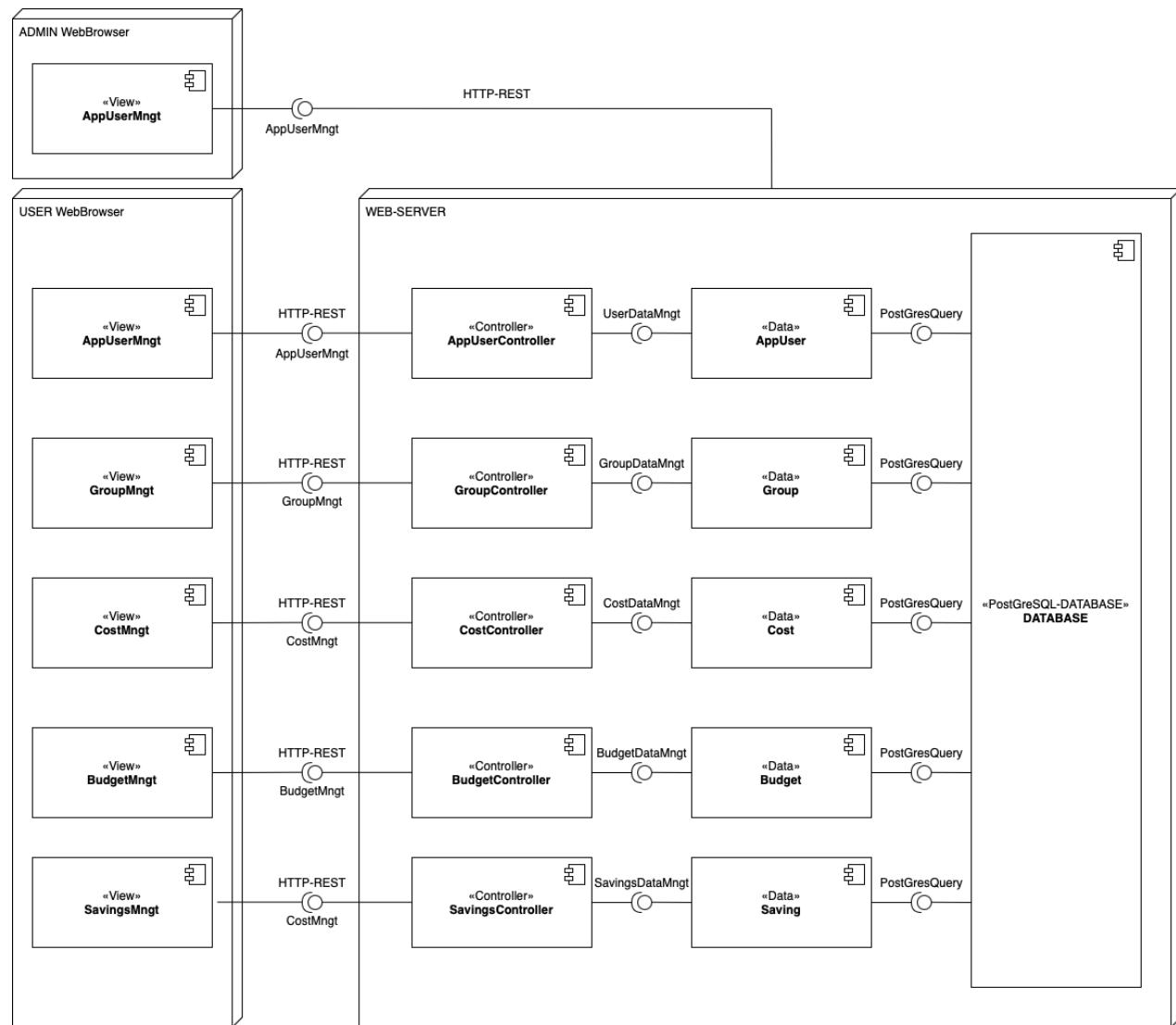


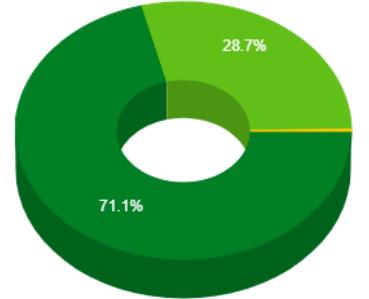
Figura 39: UML Deployment Diagram

4.7 Testing

4.7.1 Analisi Statica - CodeMR

Questa sezione documenta l'analisi statica eseguita con CodeMR per l'iterazione 3.

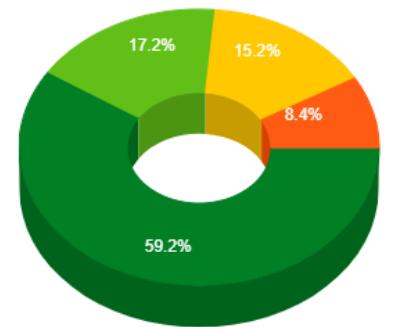
Analysis of spendly
 General Information
Total lines of code: 1324
Number of classes: 39
Number of packages: 7
Number of external packages: 44
Number of external classes: 244
Number of problematic classes: 1
Number of highly problematic classes: 0



- Complexity**
- Very High
 - High
 - Medium-high
 - Low-medium
 - Low

Figura 40: Complexity

Analysis of spendly
 General Information
Total lines of code: 1324
Number of classes: 39
Number of packages: 7
Number of external packages: 44
Number of external classes: 244
Number of problematic classes: 1
Number of highly problematic classes: 0



- Coupling**
- Very High
 - High
 - Medium-high
 - Low-medium
 - Low

Figura 41: Coupling

Detailed metric tables

- Classes with high coupling, high complexity, low cohesion (#0)
- Classes with high coupling, high complexity (#0)
- Classes with high coupling (#1)
- Classes with high complexity (#0)
- List of all classes (#39)

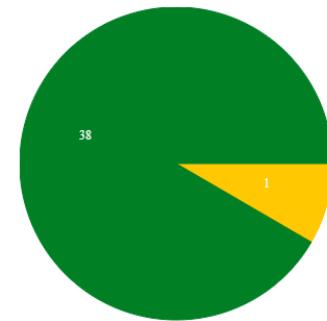


Figura 42: Problemi classi

Classes with high coupling (#1)

ID	CLASS	COUPLING	COMPLEXITY	LACK OF COHESION	SIZE	LOC	CBO	CBO APP	CBO LIB	RFC
1	AccountController	■	■	■	■	111	21	4	17	59

Figura 43: Problemi classi

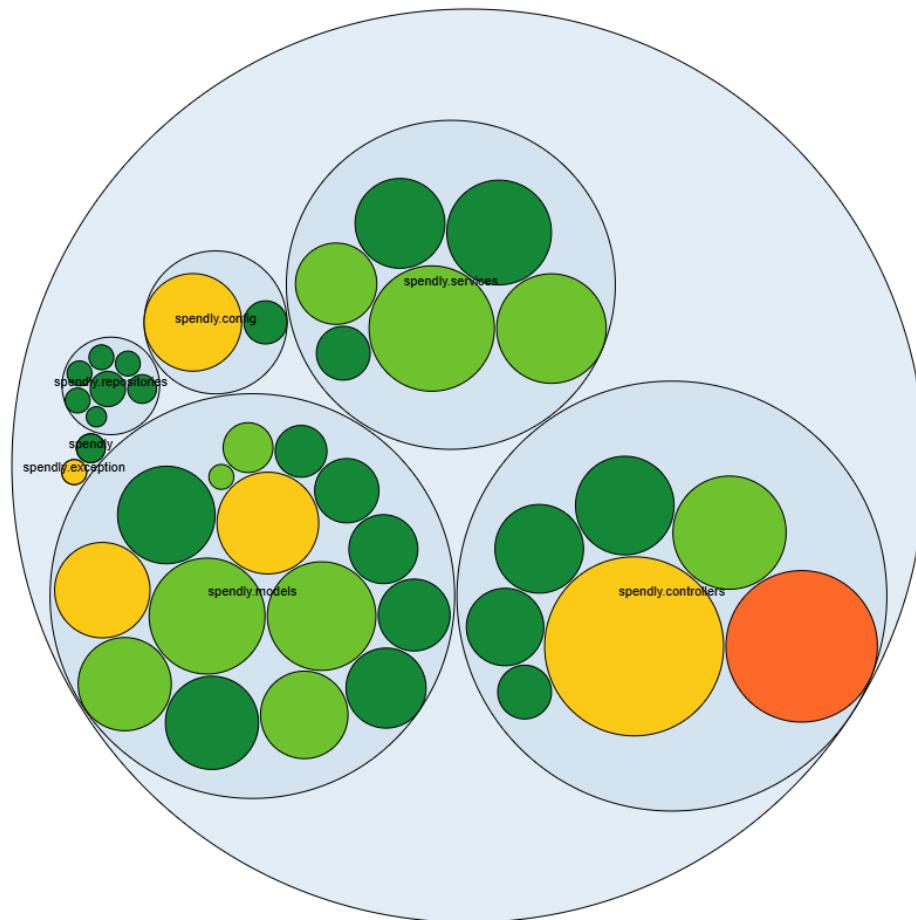


Figura 44: Struttura dei package

4.7.2 Analisi Dinamica - JUnit

Questa sezione documenta i test d'unità sviluppati per l'iterazione 3. Sono stati implementati i test riguardanti budget e risparmio.

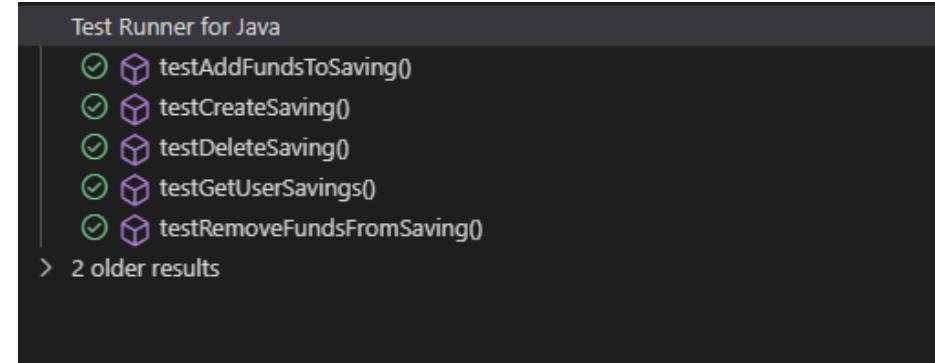


Figura 45: Test per la classe SavingService

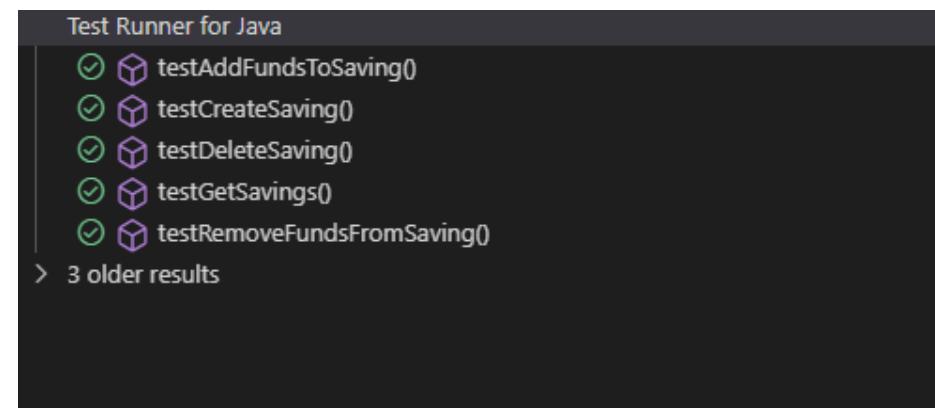


Figura 46: Test per la classe SavingController

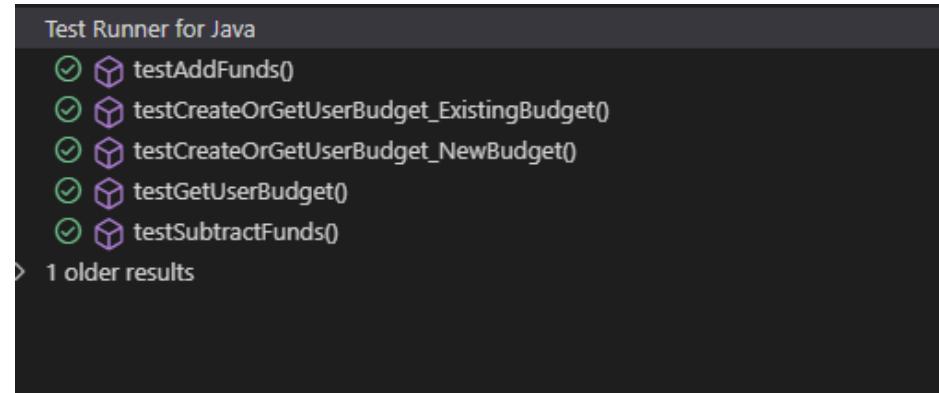


Figura 47: Test per la classe BudgetService

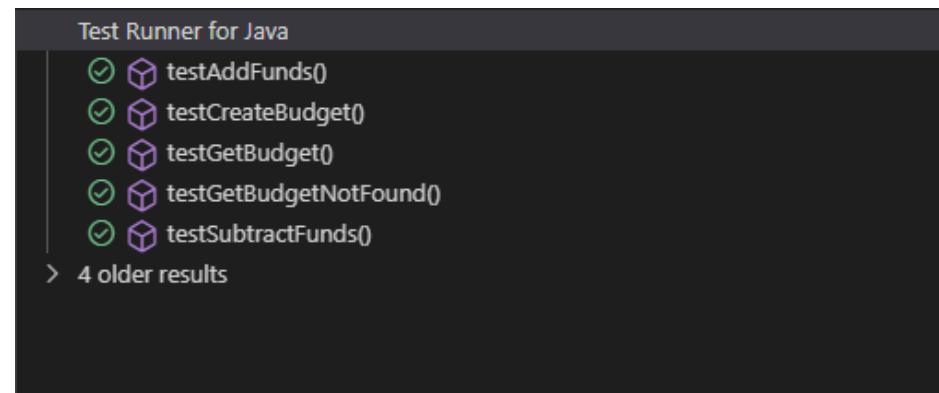


Figura 48: Test per la classe BudgetController

4.7.3 API

Questa sezione documenta il testing delle API sviluppate durante l'iterazione 3. Le API sviluppate riguardano budget e risparmio.

Creazione di un budget

- **Endpoint:** POST /api/budget/create
- **Descrizione:** Permette all'utente creare un nuovo Budget.
- **Parametri:**

- **username** (String) - Username utente.
- **amount** (double) - Valore budget.

HTTP <http://localhost:8080/api/budget/create?username=MCarr&initialAmount=200.0>

POST <http://localhost:8080/api/budget/create?username=MCarr&initialAmount=200.0>

Params • Authorization • Headers (9) Body • Scripts Settings

Query Params

Key	Value
username	MCarr
initialAmount	200.0

Body Cookies Headers (14) Test Results | ⏱

{ } JSON ▾ ▶ Preview ⚡ Visualize | ▾

```

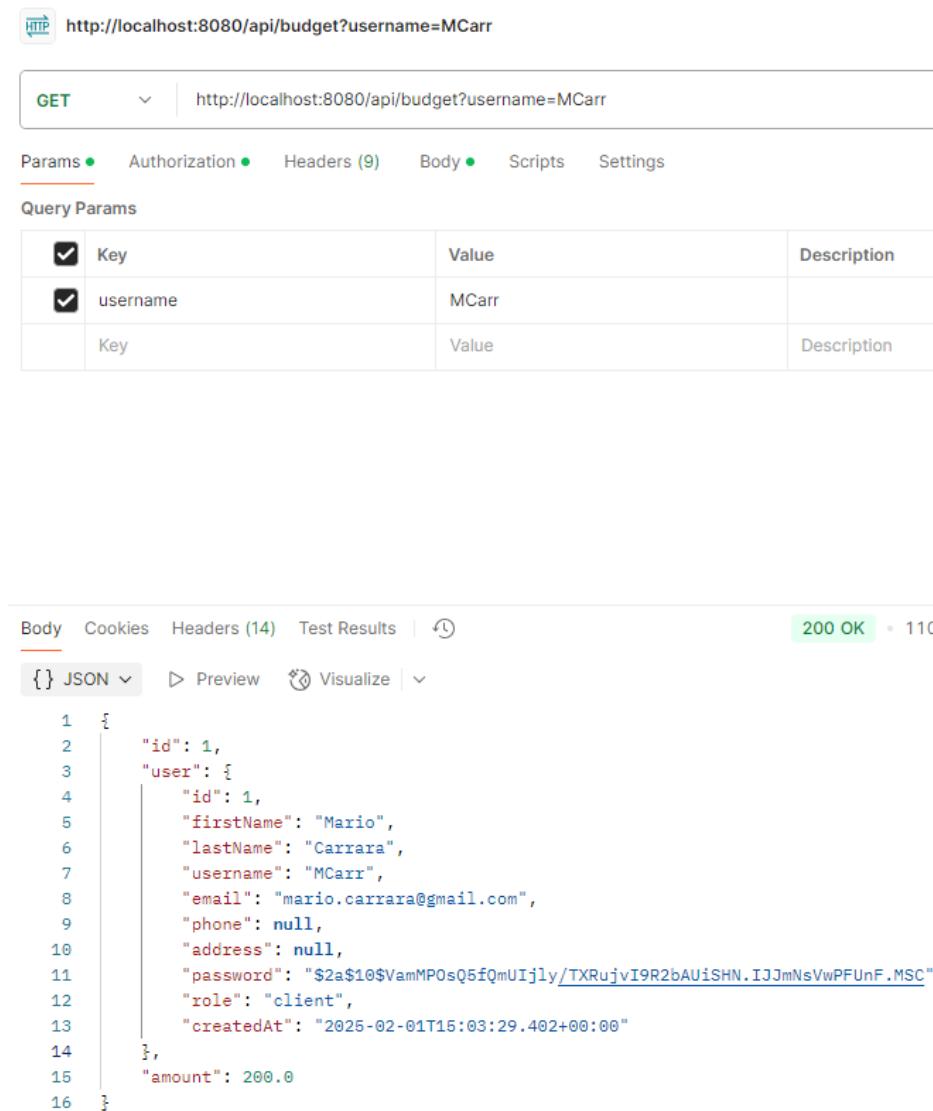
1  {
2    "id": 1,
3    "user": {
4      "id": 1,
5      "firstName": "Mario",
6      "lastName": "Carrara",
7      "username": "MCarr",
8      "email": "mario.carrara@gmail.com",
9      "phone": null,
10     "address": null,
11     "password": "$2a$10$VamMPOsQ5fQmUIjly/TXRujvI9R2bAUiSHN.IJJmNsVwPFUnF.MSC",
12     "role": "client",
13     "createdAt": "2025-02-01T15:03:29.402+00:00"
14   },
15   "amount": 200.0
16 }
```

Figura 49: API creazione budget

Visualizza budget utente

- **Endpoint:** GET /api/budget
- **Descrizione:** Permette di ottenere il budget di un utente.
- **Parametri:**

- **username** (String) - Username utente.



The screenshot shows the Postman interface with the following details:

- Method:** GET
- URL:** http://localhost:8080/api/budget?username=MCarr
- Params:** username (checkbox checked, value MCarr)
- Headers:** (9 items listed)
- Body:** (JSON response shown below)
- Status:** 200 OK (110 bytes)

```

1  {
2   "id": 1,
3   "user": {
4     "id": 1,
5     "firstName": "Mario",
6     "lastName": "Carrara",
7     "username": "MCarr",
8     "email": "mario.carrara@gmail.com",
9     "phone": null,
10    "address": null,
11    "password": "$2a$10$VamMPosQ5fQmUIjly/TXRujvI9R2bAUiSHN.IJ3mNsVwPFUnF.MSC",
12    "role": "client",
13    "createdAt": "2025-02-01T15:03:29.402+00:00"
14  },
15  "amount": 200.0
16 }

```

Figura 50: API visualizza budget di un utente

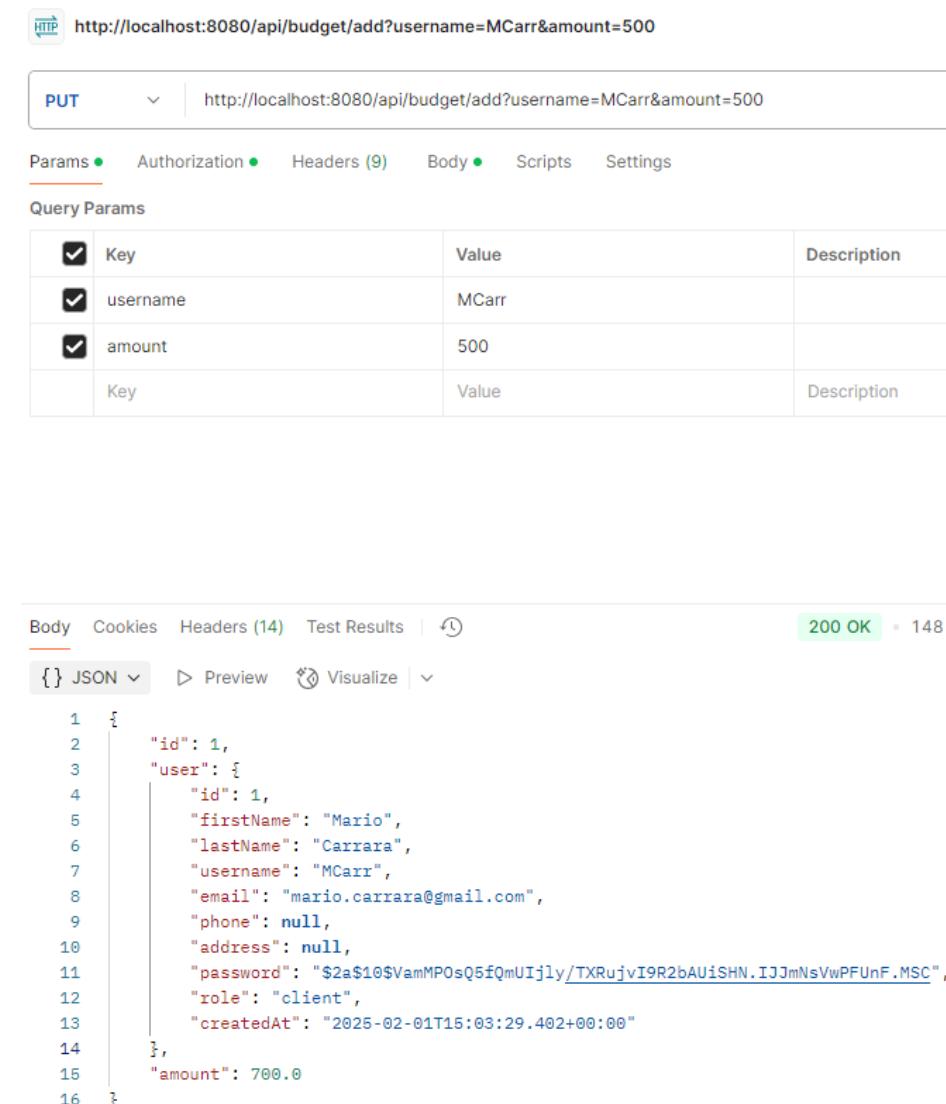
Incrementa budget

- **Endpoint:** PUT /api/budget/add

- **Descrizione:** Permette di incrementare il valore del budget

- **Parametri:**

- **username** (String) - Username utente.
- **amount** (double) - Valore da aggiungere al budget.



http://localhost:8080/api/budget/add?username=MCarr&amount=500

PUT http://localhost:8080/api/budget/add?username=MCarr&amount=500

Params • Authorization • Headers (9) Body • Scripts Settings

Query Params

Key	Value	Description
username	MCarr	
amount	500	
Key	Value	Description

Body Cookies Headers (14) Test Results 200 OK • 148 r

{ } JSON ▾ Preview Visualize

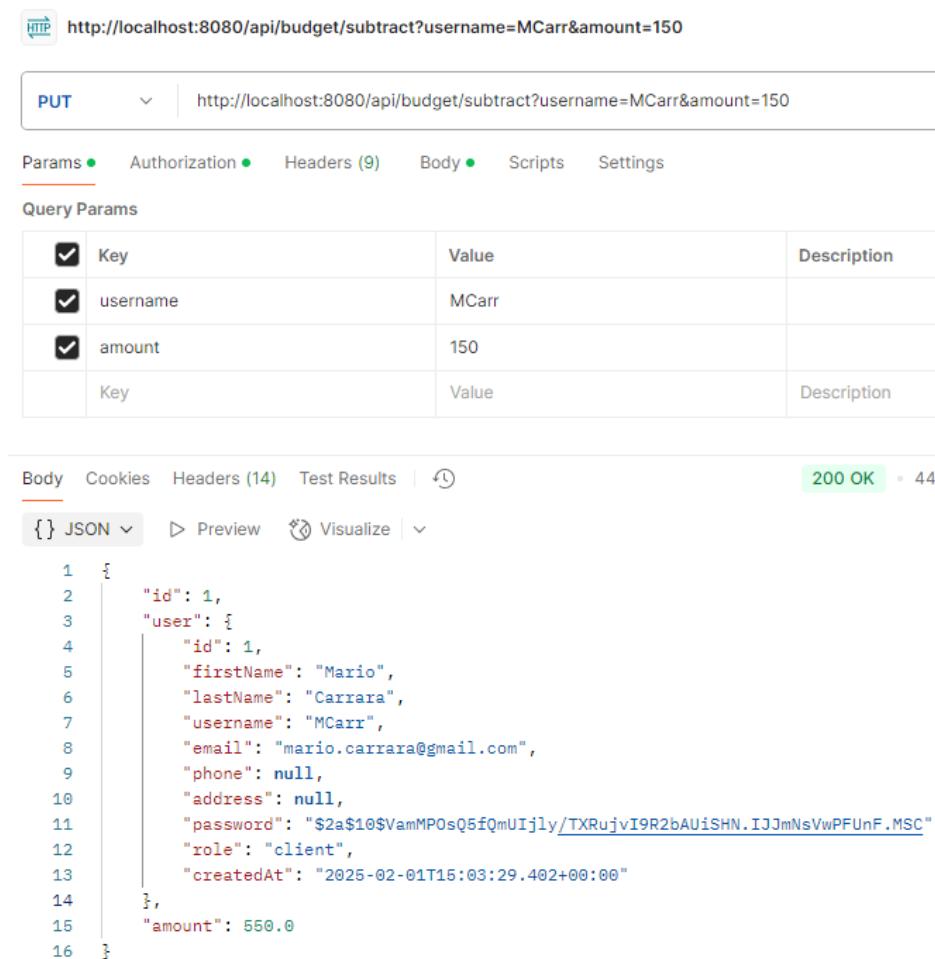
```

1  {
2   "id": 1,
3   "user": {
4     "id": 1,
5     "firstName": "Mario",
6     "lastName": "Carrara",
7     "username": "MCarr",
8     "email": "mario.carrara@gmail.com",
9     "phone": null,
10    "address": null,
11    "password": "$2a$10$VamMPOsQ5fQmUIjIy/TXRujvI9R2bAUiSHN.IJjmNsVwPFUnF.MSC",
12    "role": "client",
13    "createdAt": "2025-02-01T15:03:29.402+00:00"
14  },
15  "amount": 700.0
16 }
```

Figura 51: API per aggiungere fondi al budget

Decrementa budget

- **Endpoint:** PUT /api/budget/subtract
- **Descrizione:** Permette di decrementare il valore del budget.
- **Parametri:**
 - **username** (String) - Username utente.
 - **amount** (double) - Valore da sottrarre al budget.



HTTP <http://localhost:8080/api/budget/subtract?username=MCarr&amount=150>

PUT <http://localhost:8080/api/budget/subtract?username=MCarr&amount=150>

Params • Authorization • Headers (9) Body • Scripts • Settings

Query Params

Key	Value	Description
username	MCarr	
amount	150	
Key	Value	Description

Body Cookies Headers (14) Test Results | [Copy](#)

200 OK • 44

{ } JSON ▾ ▷ Preview ⚙ Visualize | ▾

```

1  {
2    "id": 1,
3    "user": {
4      "id": 1,
5      "firstName": "Mario",
6      "lastName": "Carrara",
7      "username": "MCarr",
8      "email": "mario.carrara@gmail.com",
9      "phone": null,
10     "address": null,
11     "password": "$2a$10$VamMPosQ5fQmUIjly/TXRujvI9R2bAUiSHN.IJ3mNsVwPFUnF.MSC",
12     "role": "client",
13     "createdAt": "2025-02-01T15:03:29.402+00:00"
14   },
15   "amount": 550.0
16 }
```

Figura 52: API per sottrarre fondi al budget

Crea risparmio

- **Endpoint:** POST /api/savings/create
- **Descrizione:** Permette di creare un piano di risparmio.
- **Parametri:**
 - `username` (String) - Username utente.
 - `name` (String) - Nome del piano di risparmio.
 - `amount` (double) - Valore del piano di risparmio.

HTTP <http://localhost:8080/api/savings/create?username=MCarr&name=Vacanza&amount=250>

POST <http://localhost:8080/api/savings/create?username=MCarr&name=Vacanza&amount=250>

Params • Authorization • Headers (9) Body • Scripts Settings

Query Params

Key	Value	Description
username	MCarr	
name	Vacanza	
amount	250	

Body Cookies Headers (14) Test Results 200 OK 176 i

{ } JSON ▾ Preview ⏪ Visualize ▾

```

1  {
2   "id": 1,
3   "name": "Vacanza",
4   "amount": 250.0,
5   "user": {
6     "id": 1,
7     "firstName": "Mario",
8     "lastName": "Carrara",
9     "username": "MCarr",
10    "email": "mario.carrara@gmail.com",
11    "phone": null,
12    "address": null,
13    "password": "$2a$10$VamMPOsQ5fQmUIjly/TXRujvI9R2bAUiSHN.IJJmNsVwPFUnF.MSC",
14    "role": "client",
15    "createdAt": "2025-02-01T15:03:29.402+00:00"
16  }
17 }
```

Figura 53: API per creare un piano di risparmio

Visualizza risparmi utente

- **Endpoint:** GET /api/savings
- **Descrizione:** Permette di ottenere i piani di risparmio dell'utente.
- **Parametri:**
 - **username** (String) - Username utente.

HTTP <http://localhost:8080/api/savings?username=MCarr>

GET <http://localhost:8080/api/savings?username=MCarr>

Params • Authorization • Headers (9) • Body • Scripts • Settings

Query Params

<input checked="" type="checkbox"/> Key	Value	Description
<input checked="" type="checkbox"/> username	MCarr	
	Value	Description

Body Cookies Headers (14) Test Results | 

200 OK • 43 ms • 1

[] JSON ▾ ▷ Preview ⚡ Visualize | ▾

```

1  [
2    {
3      "id": 1,
4      "name": "Vacanza",
5      "amount": 250.0,
6      "user": {
7        "id": 1,
8        "firstName": "Mario",
9        "lastName": "Carrara",
10       "username": "MCarr",
11       "email": "mario.carrara@gmail.com",
12       "phone": null,
13       "address": null,
14       "password": "$2a$10$VamMPosQ5fQmUIjly/TXRujvI9R2bAUiSHN.IJjmNsVwPFUnF.MSC",
15       "role": "client",
16       "createdAt": "2025-02-01T15:03:29.402+00:00"
17     }
18   },
19   {
20     "id": 2,
21     "name": "Sport",
22     "amount": 50.0,

```

Figura 54: API per ottenere i piani di risparmio dell'utente

Aggiungi fondo al risparmio

- **Endpoint:** PUT /api/savings/addFunds
- **Descrizione:** Permette di aggiungere fondi al piano di risparmio dell'utente.
- **Parametri:**
 - **username** (String) - Username utente.

- **savingId** (Long) - Id del risparmio.
- **amount** (double) - Valore da aggiungere al piano di risparmio dell’utente.

HTTP <http://localhost:8080/api/savings/addFunds?username=MCarr&savingId=1&amount=20>

PUT <http://localhost:8080/api/savings/addFunds?username=MCarr&savingId=1&amount=20>

Params	Authorization	Headers (9)	Body	Scripts	Settings
<input checked="" type="checkbox"/> Key			Value		Description
<input checked="" type="checkbox"/> username			MCarr		
<input checked="" type="checkbox"/> savingId			1		
<input checked="" type="checkbox"/> amount			20		
			Key	Value	Description

Body Cookies Headers (14) Test Results |

200 OK 50 ms

{ } JSON ▾ ▶ Preview ⚡ Visualize ▾

```

1  {
2    "id": 1,
3    "name": "Vacanza",
4    "amount": 270.0,
5    "user": [
6      {
7        "id": 1,
8        "firstName": "Mario",
9        "lastName": "Carrara",
10       "username": "MCarr",
11       "email": "mario.carrara@gmail.com",
12       "phone": null,
13       "address": null,
14       "password": "$2a$10$VamMPOsQ5fQmUIjly/TXRujvI9R2bAUiSHN.IJJmNsVwPFUnF.MSC",
15       "role": "client",
16       "createdAt": "2025-02-01T15:03:29.402+00:00"
17     }
  
```

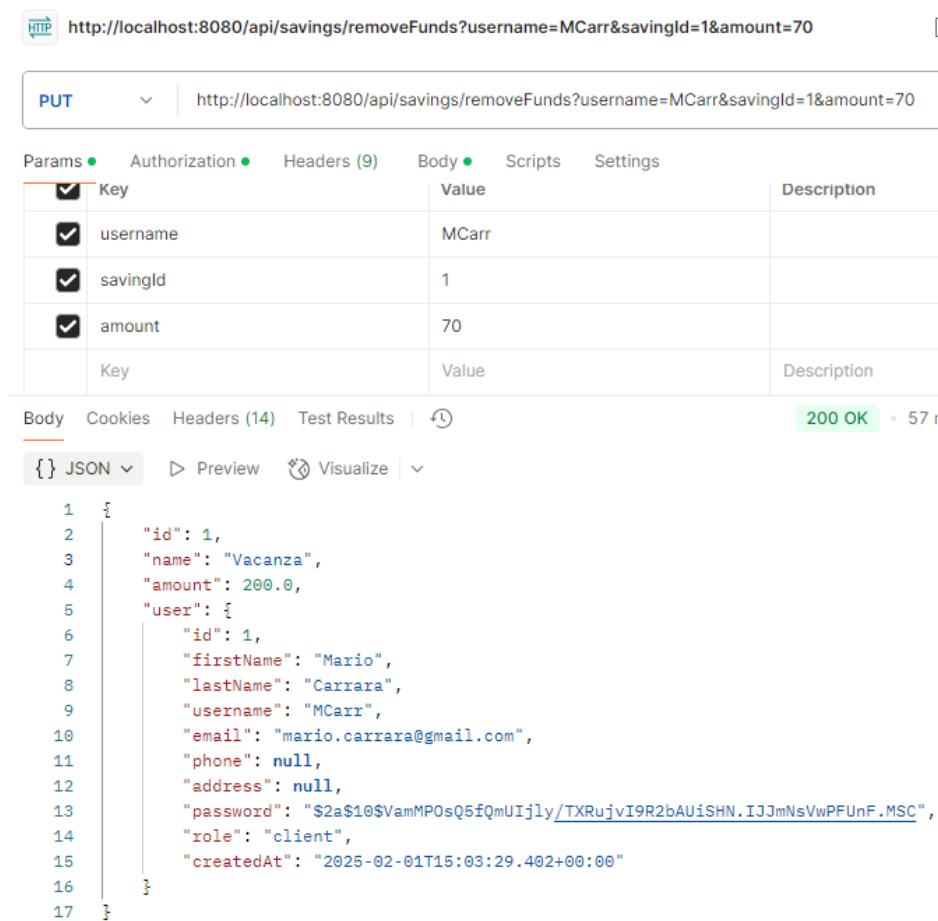
Figura 55: API per aggiungere fondi al piano di risparmio dell’utente

Rimuovi fondo al risparmio

- **Endpoint:** PUT /api/savings/removeFunds
- **Descrizione:** Permette di togliere fondi al piano di risparmio dell’utente.

- **Parametri:**

- **username** (String) - Username utente.
- **savingId** (Long) - Id del piano di risparmio.
- **amount** (double) - Valore da togliere al piano di risparmio.



The screenshot shows a POST request to `http://localhost:8080/api/savings/removeFunds?username=MCarr&savingId=1&amount=70`. The request body is a JSON object representing a savings plan:

```

1  {
2    "id": 1,
3    "name": "Vacanza",
4    "amount": 200.0,
5    "user": {
6      "id": 1,
7      "firstName": "Mario",
8      "lastName": "Carrara",
9      "username": "MCarr",
10     "email": "mario.carrara@gmail.com",
11     "phone": null,
12     "address": null,
13     "password": "$2a$10$VamMP0sQ5fQmUIjly/TXRujvI9R2bAUiSHN.IJJmNsVwPFUnF.MSC",
14     "role": "client",
15     "createdAt": "2025-02-01T15:03:29.402+00:00"
16   }
17 }

```

Figura 56: API per togliere fondi al piano di risparmio dell'utente

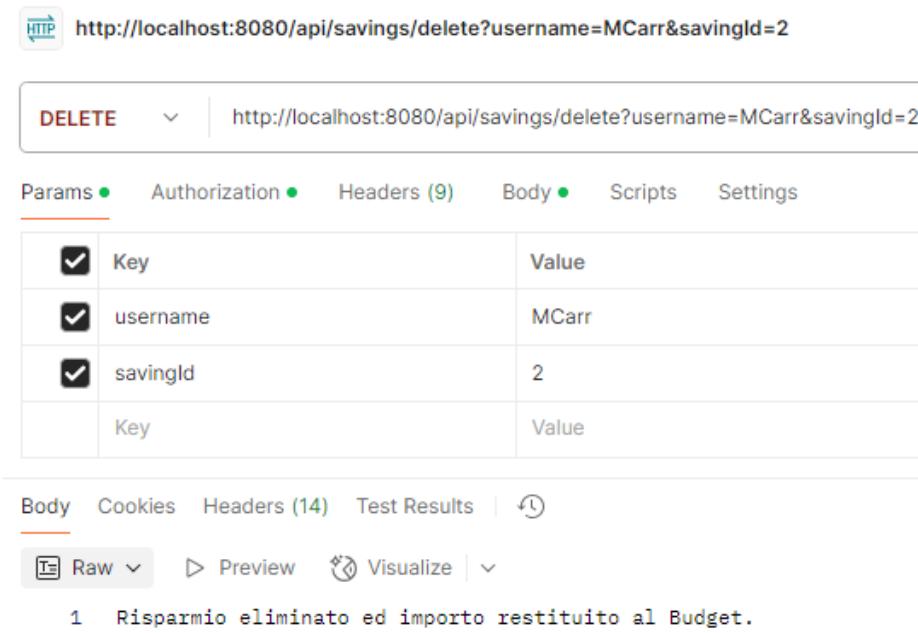
Elimina risparmio

- **Endpoint:** `DELETE /api/savings/delete`

- **Descrizione:** Permette di eliminare un piano di risparmio.

- **Parametri:**

- **username** (String) - Username utente.
- **savingId** (Long) - Id del piano di risparmio.



HTTP <http://localhost:8080/api/savings/delete?username=MCarr&savingId=2>

DELETE <http://localhost:8080/api/savings/delete?username=MCarr&savingId=2>

Params	Authorization	Headers (9)	Body	Scripts	Settings
<input checked="" type="checkbox"/> Key					
<input checked="" type="checkbox"/> username			MCarr		
<input checked="" type="checkbox"/> savingId			2		
			Key		Value

Body Cookies Headers (14) Test Results | 

 Raw ▾  Preview  Visualize | 

```
1 Risparmio eliminato ed importo restituito al Budget.
```

Figura 57: API per eliminare un piano di risparmio