Gestione Membri

Sotto-funzionalità	Sotto-funzionalità	Legame	Informazioni
Assegnazione / Rimozione Ruolo Lavoratore	Rimozione Membro	Il ruolo di Lavoratore non può essere assegnato o rimosso se il Membro è stato rimosso	Email Membro
Rimozione Ruolo Lavoratore	Assegnazione Ruolo Lavoratore	Il ruolo di Lavoratore non può essere rimosso se non è stato assegnato	

Gestione Spese

Sotto-funzionalità	Sotto-funzionalità	Legame	Informazioni
Eliminazione Spesa	Creazione Spesa	Una Spesa non può essere eliminata se non è stato creata	Spesa

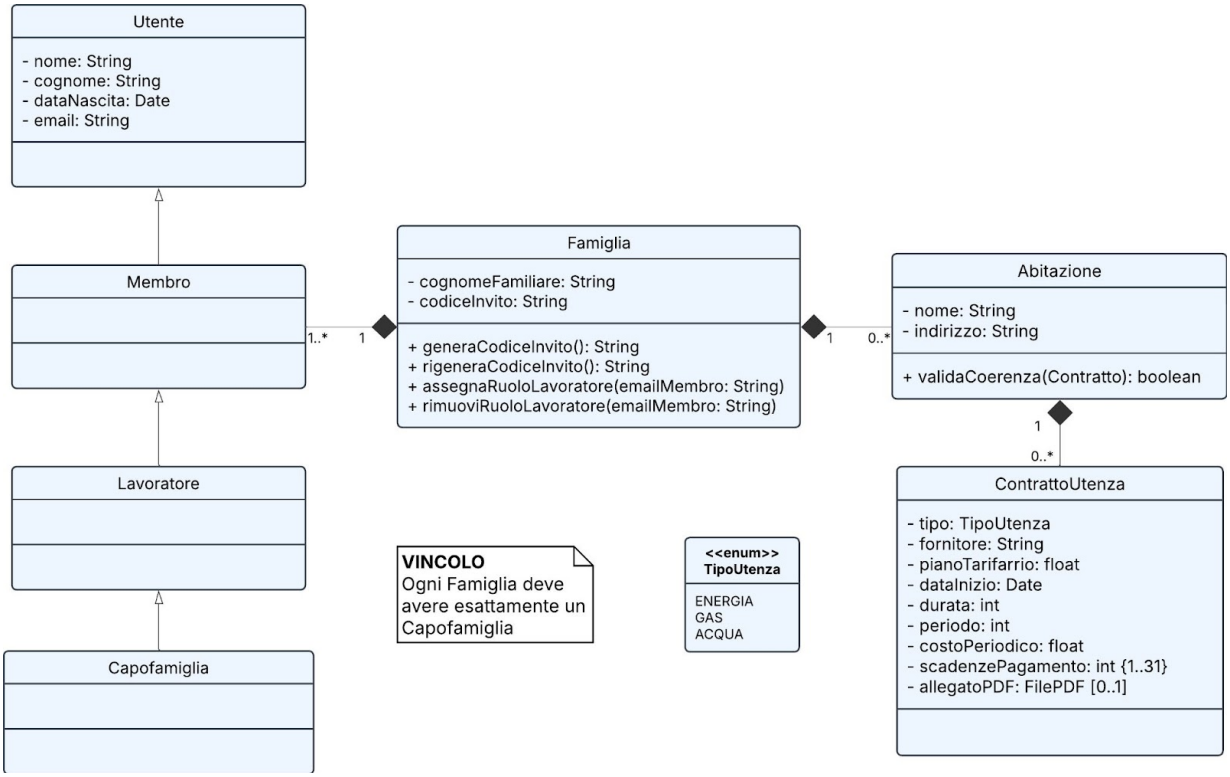
Gestione Introiti

Sotto-funzionalità	Sotto-funzionalità	Legame	Informazioni
Eliminazione Introito	Creazione Introito	Un Introito non può essere eliminato se non è stato creato	Introito

Modello del dominio

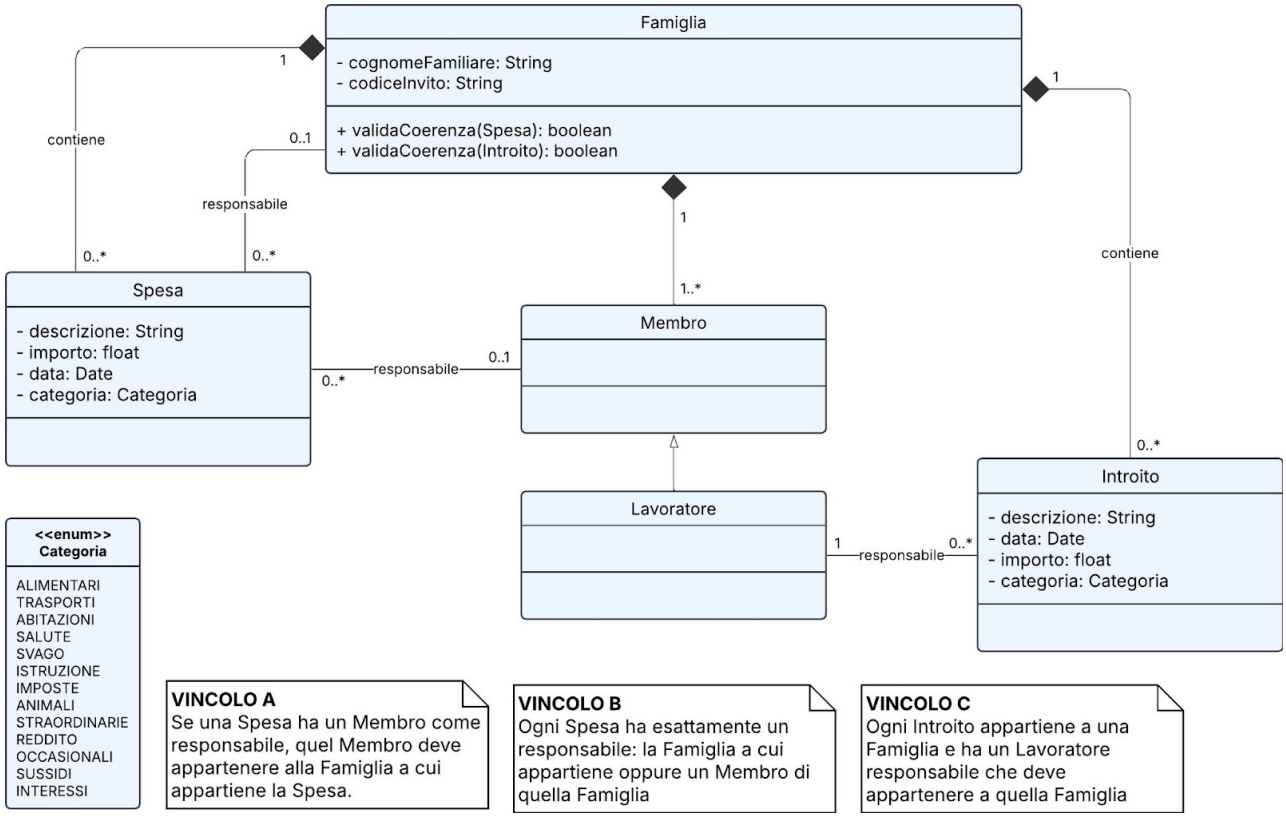
Famiglia

Di seguito viene mostrato il modello per la **gestione della Famiglia e delle Abitazioni.**



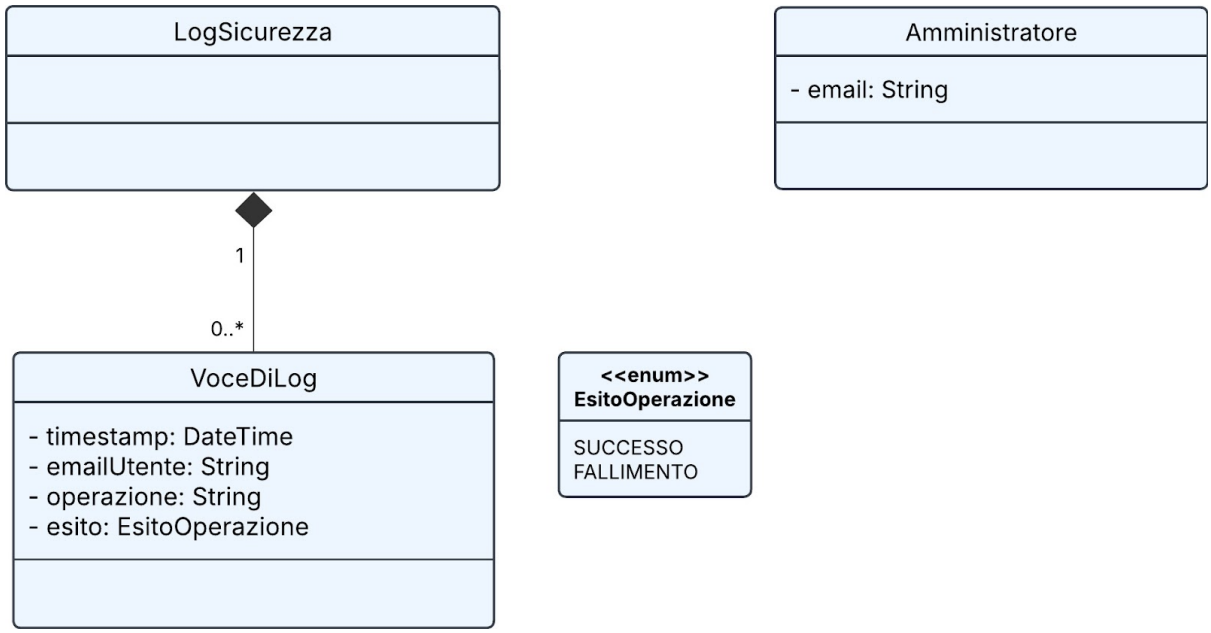
Spesa e Introiti

Di seguito viene mostrato il modello per la **gestione delle Spese e degli Introiti**.



Amministratore

Di seguito viene mostrato il modello per la **gestione del log da parte dell'Amministratore**.



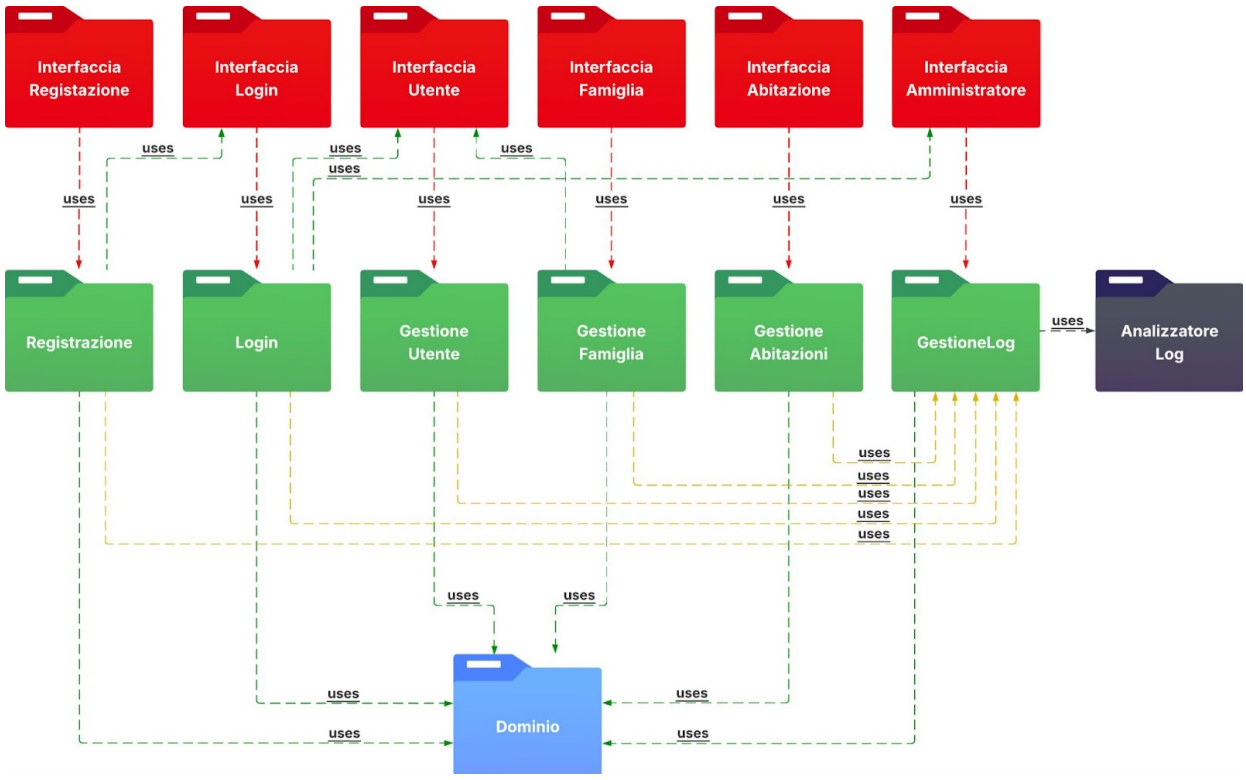
Architettura logica

Struttura

Diagramma dei package

La struttura dei package segue il pattern architetturale Entity-Control-Boundary, nel quale:

- Gli attori definiti durante l'analisi dei requisiti interagiscono solo con i package di interfaccia (boundary), evidenziati in rosso
- Il modello (model), mostrato in azzurro, descrive le entità coinvolte.
- I controller, contrassegnati in verde, gestiscono la logica di business e fanno da ponte tra l'interfaccia utente e il modello

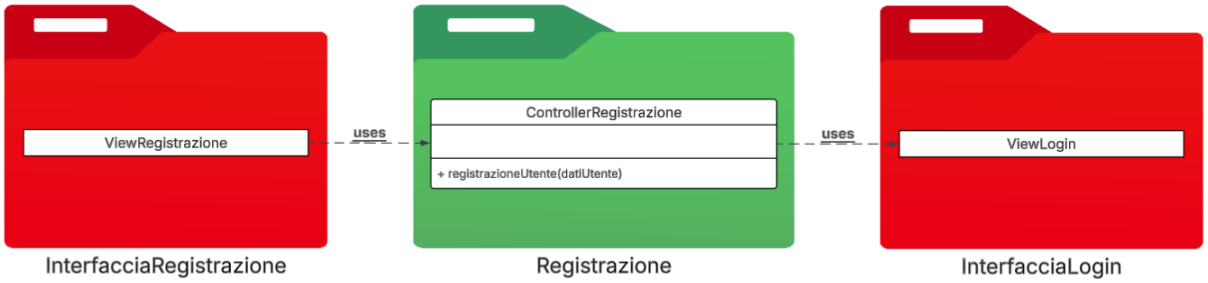


I collegamenti in giallo indicano le dipendenze verso il sistema di logging, per rendere il grafo più leggibile.

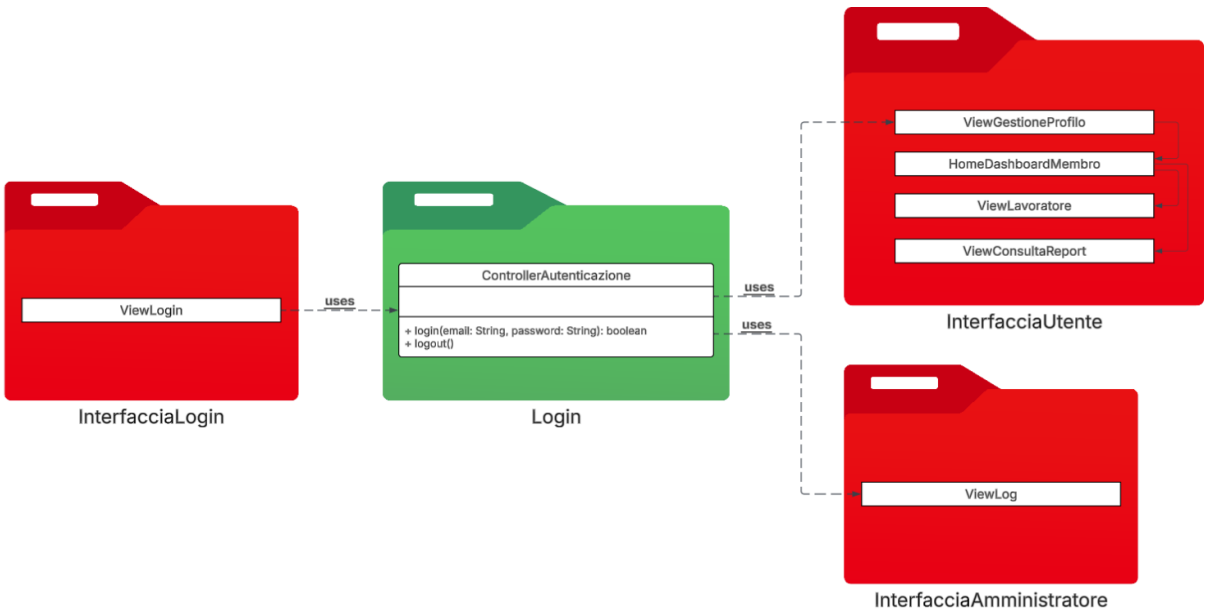
Diagramma delle classi

È stato evidenziato precedentemente come ogni controller si collega al controller dei log. Per migliorare la leggibilità dei successivi diagrammi abbiamo omesso tutti i collegamenti verso di esso.

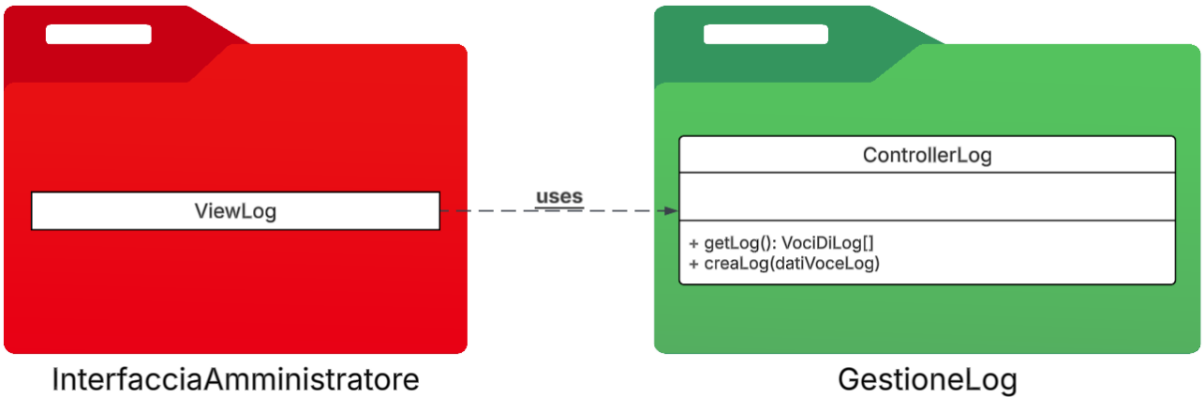
Login



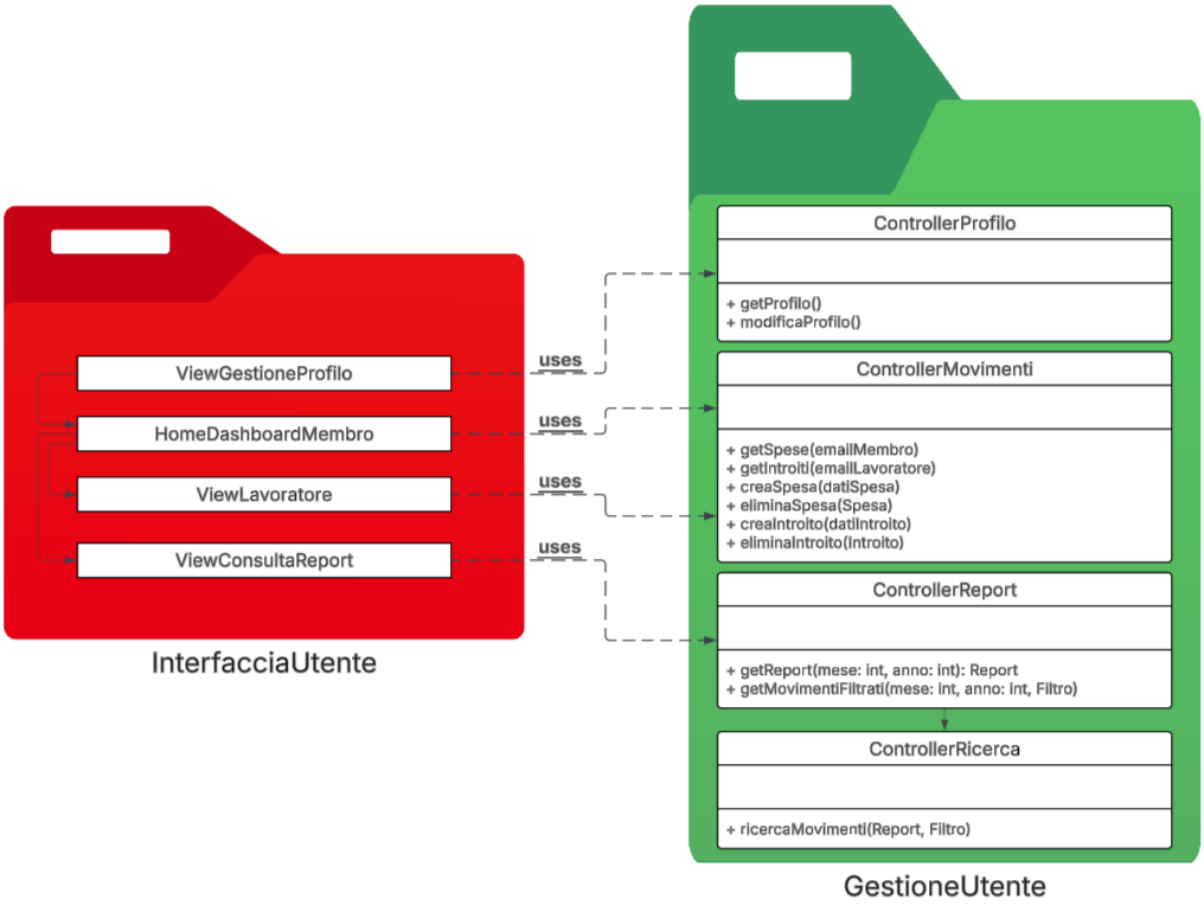
Registrazione



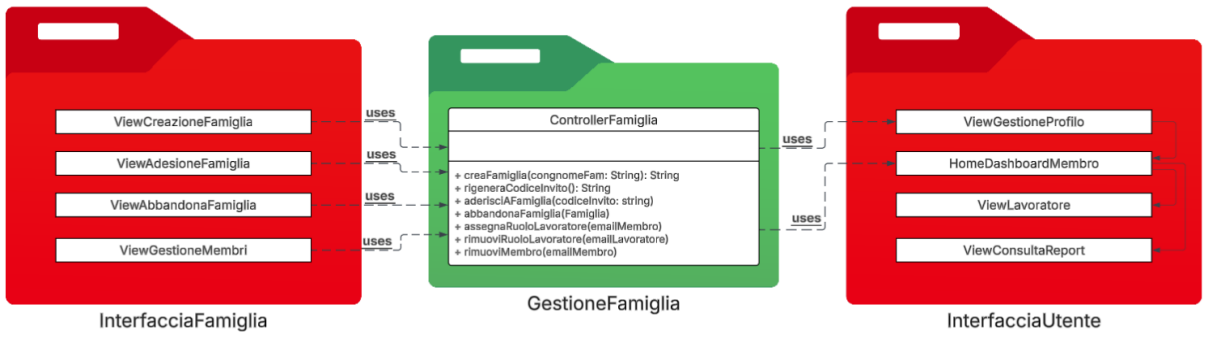
Amministratore



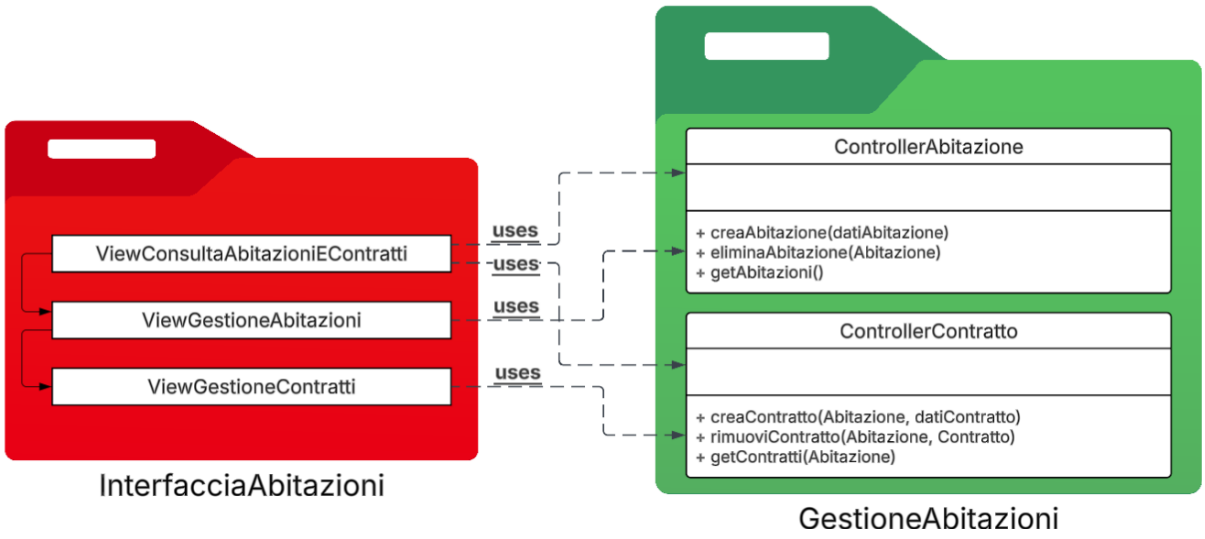
Utenti



Famiglia



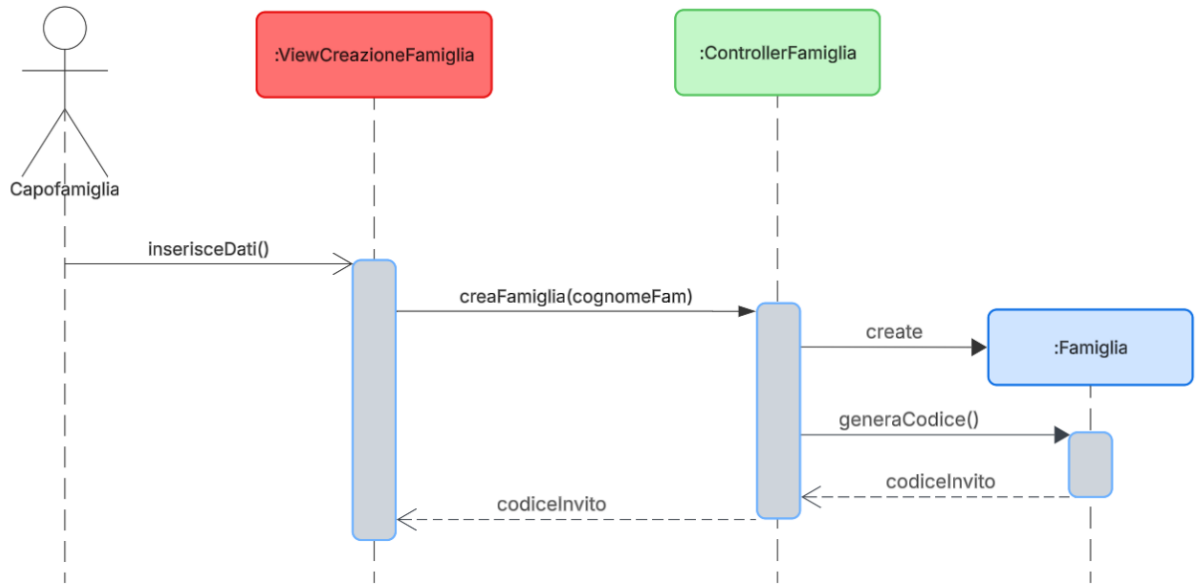
Abitazione



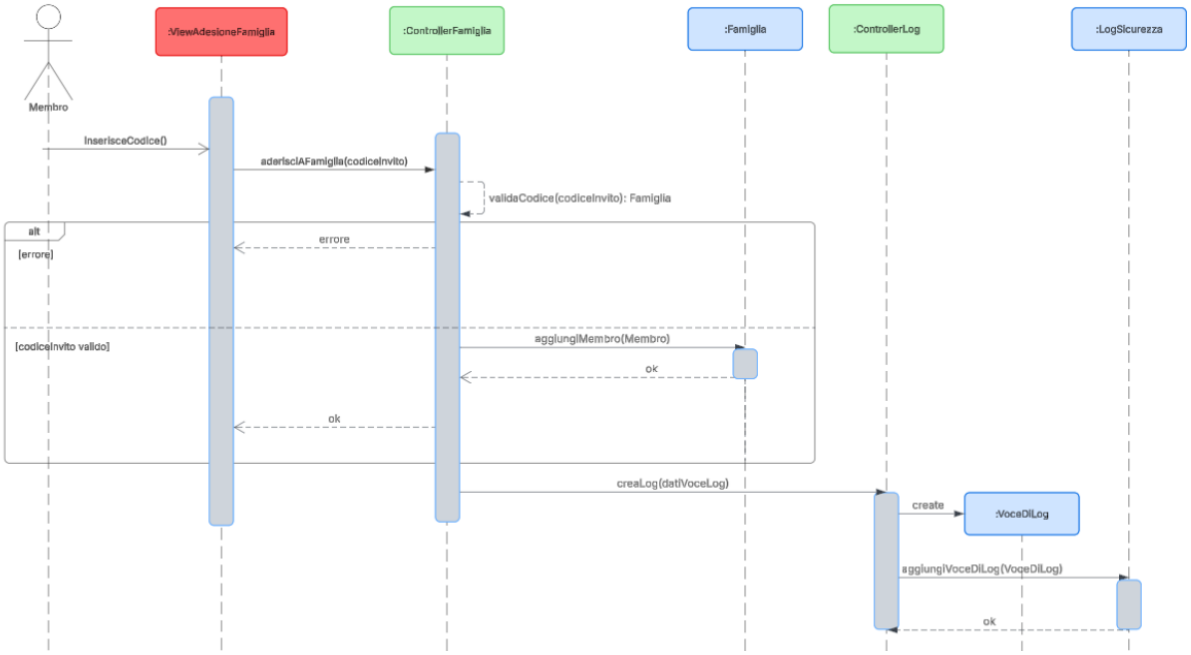
Interazioni

Abbiamo sottinteso che ad ogni interazione con il sistema verrà inserita una nuova voce di log apposita.

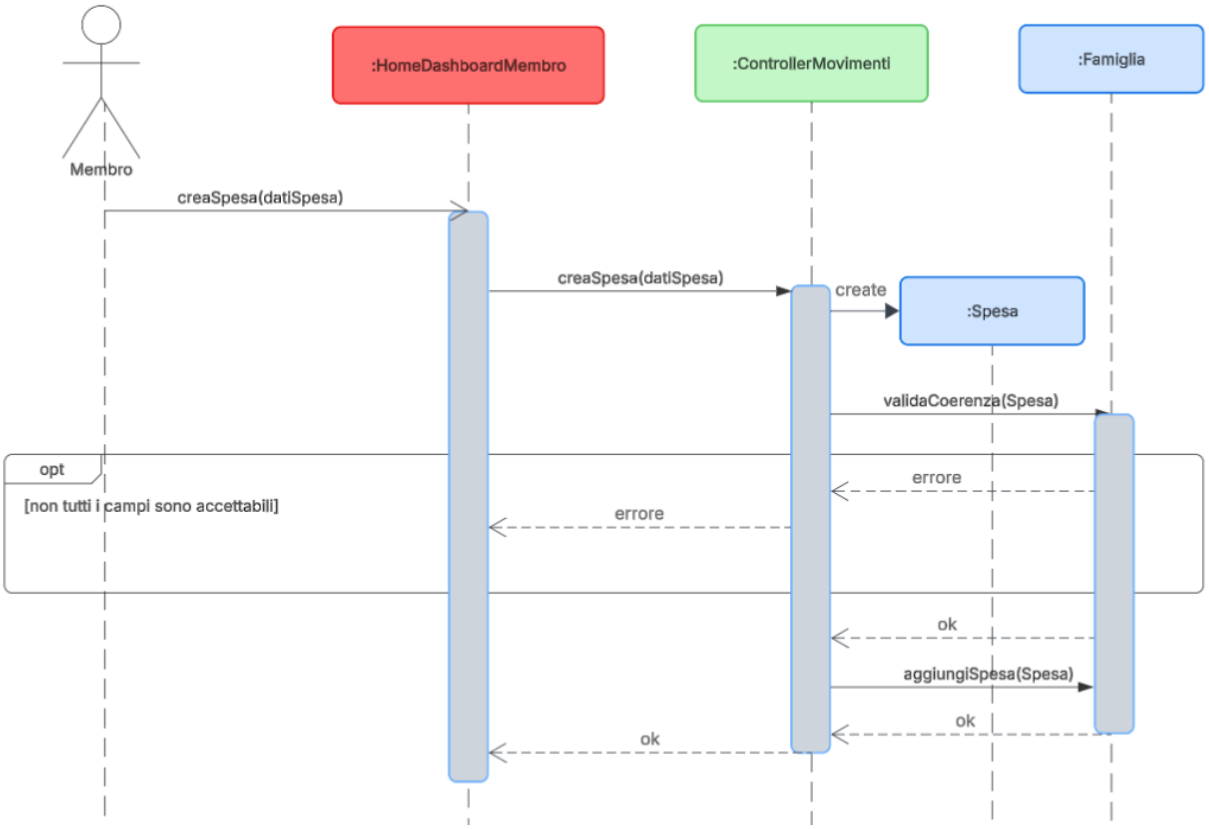
Capofamiglia crea una Famiglia



Membro aderisce a una Famiglia

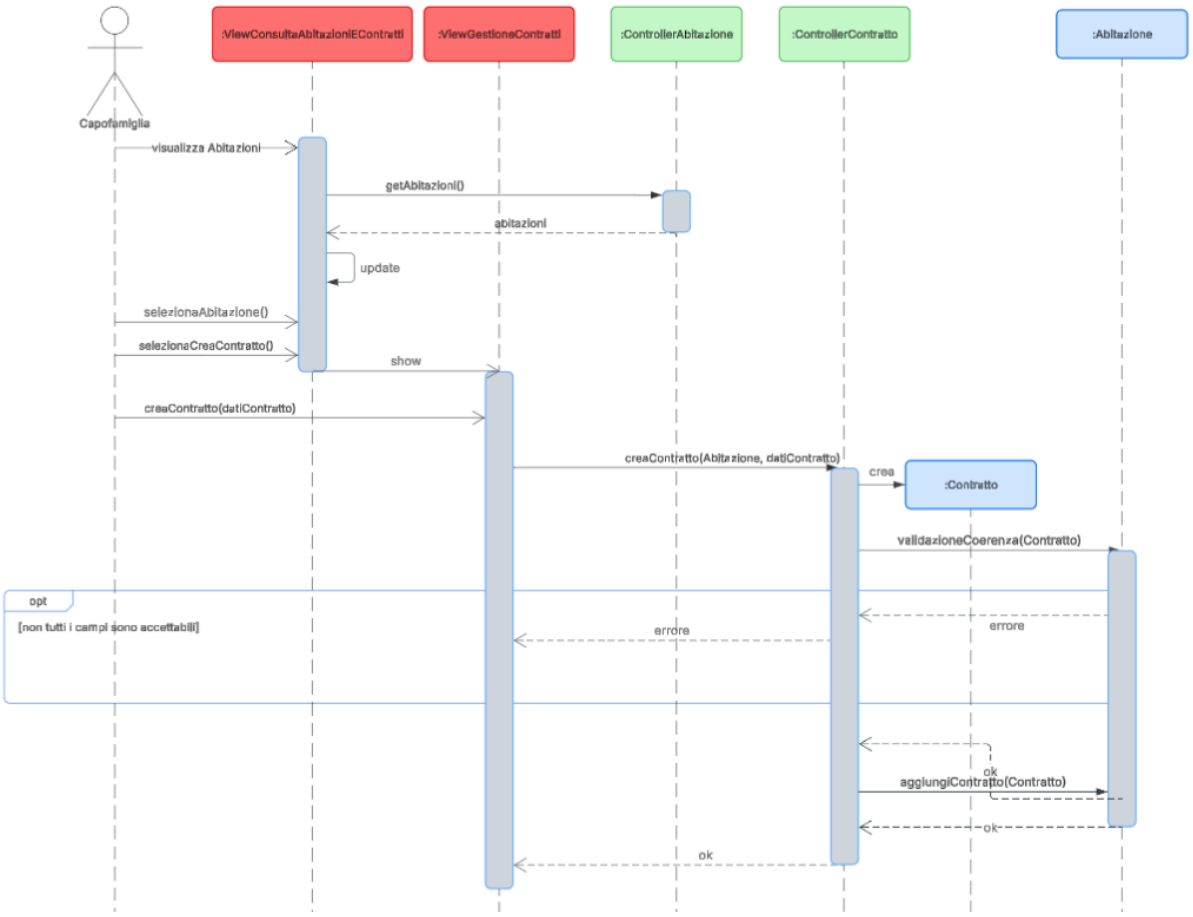


Membro crea una Spesa



Caso analogo per aggiunta Introito.

Capofamiglia crea un Contratto



Piano di lavoro

Composizione Team

Nome Team	Composizione
Team Progettazione	Marziano, Rapp, Santi
Team Sviluppo	Marziano, Rapp, Santi
Team DB	Marziano, Rapp, Santi
Team Sicurezza	Marziano, Rapp, Santi
Team Frontend	Marziano, Rapp, Santi

Package e Responsabilità

Package	Progettazione	Sviluppo
Dominio	• Team Progettazione • Team DB	• Team Sviluppo
Log	• Team Progettazione • Team Sicurezza	• Team Sviluppo
Registazione	• Team Progettazione • Team Sicurezza • Team DB	• Team Sviluppo • Team Sicurezza
Login	• Team Progettazione • Team Sicurezza	• Team Sviluppo • Team Sicurezza
Gestione Utente	• Team Progettazione • Team DB	• Team Sviluppo
Gestione Famiglia	• Team Progettazione • Team DB	• Team Sviluppo
Gestione Abitazioni	• Team Progettazione • Team DB	• Team Sviluppo
Interfaccia Registrazione	• Team Progettazione • Team Frontend • Team Sicurezza	• Team Sviluppo • Team Sicurezza
Interfaccia Login	• Team Progettazione • Team Frontend	• Team Sviluppo • Team Sicurezza

Package	Progettazione	Sviluppo
	• Team Sicurezza	
Interfaccia Utente	• Team Progettazione • Team Frontend	• Team Sviluppo
Interfaccia Famiglia	• Team Progettazione • Team Frontend	• Team Sviluppo
Interfaccia Abitazione	• Team Progettazione • Team Frontend	• Team Sviluppo
Interfaccia Amministratore	• Team Progettazione • Team Frontend	• Team Sviluppo

Cronologia di Sviluppo

Periodo	Fase
Novembre 2025	Primo prototipo.
Dicembre 2025	Secondo prototipo.
Febbraio 2026	Prima versione con tutte le funzionalità. Test di integrazione con sistemi esterni e test di usabilità ed esperienza utente.
Aprile 2026	Prima versione beta. Accesso previa pre-registrazione ad un numero limitato di utenti.
Maggio 2026	Rilascio al pubblico.

Il primo prototipo dovrà implementare Registrazione e Login, oltre alle principali funzionalità relative alla Gestione Famiglie:

- Creazione e modifica di una Famiglia
- Gestione membri e inviti
- Visualizzazione Dashboard base
- Aggiunta Spese ed Introiti

Al contrario, non saranno implementati:

- i sistemi di generazione report avanzati e l'esportazione dati
- l'interazione con il sistema esterno che si occupa di analizzare i log
- le funzionalità relative alla validazione e registrazione dei fornitori

Piano del Collaudo

Collaudo login

```
const { describe, it, beforeEach, afterEach } = require('mocha');
const { expect } = require('chai');
const sinon = require('sinon');
const bcrypt = require('bcrypt');
const UtenteRegistrato = require('../models/UtenteRegistrato');
const db = require('../config/database');
describe('TestUtenteRegistrato', () => {
    let utente;
    let dbStub;
    beforeEach(() => {
        dbStub = sinon.stub(db, 'query');
        utente = new UtenteRegistrato({
            nome: 'Mario',
            cognome: 'Rossi',
            username: 'mario_rossi',
            email: 'mario.rossi@email.com',
            dataNascita: '1990-05-15',
            password: 'SecurePass123!'
        });
    });
    afterEach(() => {
        dbStub.restore();
    });
    describe('testGetter', () => {
        it('dovrebbe restituire i valori corretti dei getter', () => {
            expect(utente.getNome()).to.equal('Mario');
            expect(utente.getCognome()).to.equal('Rossi');
            expect(utente.getUsername()).to.equal('mario_rossi');
            expect(utente.getEmail()).to.equal('mario.rossi@email.com');
            expect(utente.getDataNascita()).to.equal('1990-05-15');
        });
    });
    describe('testSetter', () => {
        it('dovrebbe aggiornare correttamente i valori', () => {
            utente.setNome('Giovanni');
            expect(utente.getNome()).to.equal('Giovanni');

            utente.setCognome('Bianchi');
            expect(utente.getCognome()).to.equal('Bianchi');
            utente.setEmail('giovanni.bianchi@email.com');
            expect(utente.getEmail()).to.equal('giovanni.bianchi@email.com');
        });
    });
```

Collaudo della famiglia

```
    describe('testCreazioneFamiglia', () => {
        it('dovrebbe creare una nuova famiglia con abitazione iniziale', async () => {
            transactionStub.callsFake(async (callback) => {
                const trx = {
                    query: sinon.stub().resolves({ insertId: 1 }),
                    commit: sinon.stub().resolves(),
                    rollback: sinon.stub().resolves()
                };
                await callback(trx);
                return { famigliaId: 1, codiceInvito: 'ABC123' };
            });

            const result = await famiglia.crea();

            expect(result).to.be.true;
            expect(famiglia.getCodiceInvito()).to.match(/^[A-Za-z0-9]{5}$/);
            expect(famiglia.getAbitazioni()).to.have.lengthOf(1);
        });

        it('dovrebbe creare categorie di spesa predefinite', async () => {
            const categorieBase = ['Alimentari', 'Utenze', 'Trasporti', 'Salute', 'Svago'];

            transactionStub.callsFake(async (callback) => {
                const trx = {
                    query: sinon.stub().resolves({ insertId: 1 }),
                    commit: sinon.stub().resolves(),
                    rollback: sinon.stub().resolves()
                };
                await callback(trx);
                return true;
            });

            await famiglia.crea();

            expect(famiglia.getCategorieSpesa()).to.deep.equal(categorieBase);
        });
    });

    describe('testGestioneMembri', () => {
        it('dovrebbe approvare richiesta di adesione', async () => {
            dbStub.onFirstCall().resolves([{
                id: 2,
                utente_id: 5,
                stato: 'in_attesa'
            }]);
            dbStub.onSecondCall().resolves({ affectedRows: 1 });

            const result = await famiglia.approvaAdesione(2);

            expect(result).to.be.true;
            expect(dbStub.calledTwice).to.be.true;
        });

        it('dovrebbe assegnare ruolo lavoratore a membro', async () => {
            dbStub.resolves({ affectedRows: 1 });

            const result = await famiglia.assegnaRuoloLavoratore(3);

            expect(result).to.be.true;
            expect(dbStub.firstCall.args[0]).to.include('UPDATE membri SET ruolo = ?');
            expect(dbStub.firstCall.args[1]).to.deep.equal(['lavoratore', 3, 1]);
        });

        it('dovrebbe rimuovere membro dalla famiglia', async () => {
            transactionStub.callsFake(async (callback) => {
                const trx = {
                    query: sinon.stub().resolves({ affectedRows: 1 }),
                    commit: sinon.stub().resolves(),
                    rollback: sinon.stub().resolves()
                };
                await callback(trx);
                return true;
            });

            const result = await famiglia.rimuoviMembro(4);

            expect(result).to.be.true;
        });
    });

    describe('testCodiceInvito', () => {
        it('dovrebbe generare codice invito univoco', () => {
            const codice = famiglia.generaCodiceInvito();

            expect(codice).to.match(/^[A-Za-z0-9]{5}$/);
            expect(codice).to.have.lengthOf(5);
        });
});
```

Collaudo Spese

```
describe('testCreazione', () => {
        it('dovrebbe creare una nuova spesa', async () => {
            dbStub.resolves({ insertId: 100 });

            const result = await spesa.crea();

            expect(result).to.be.true;
            expect(spesa.getId()).to.equal(100);
            expect(dbStub.firstCall.args[0]).to.include('INSERT INTO spese');
        });

        it('dovrebbe validare importo positivo', () => {
            spesa.setImporto(-50);
            const valid = spesa.validaCoerenza();

            expect(valid).to.be.false;
            expect(spesa.getErrore()).to.equal('Importo deve essere positivo');
        });

        it('dovrebbe associare spesa ad abitazione', async () => {
            spesa.setAbitazioneId(3);
            dbStub.resolves({ insertId: 101 });

            await spesa.crea();

            expect(dbStub.firstCall.args[1]).to.include(3);
        });
    });

    describe('testModifica', () => {
        it('dovrebbe modificare spesa esistente', async () => {
            spesa.setId(100);
            spesa.setDescrizione('Spesa aggiornata');
            spesa.setImporto(200);

            dbStub.resolves({ affectedRows: 1 });

            const result = await spesa.modifica();

            expect(result).to.be.true;
            expect(dbStub.firstCall.args[0]).to.include('UPDATE spese SET');
        });

        it('solo capofamiglia può modificare spese altrui', async () => {
            const capofamiglia = {
                id: 1,
                famiglia_id: 1,
                ruolo: 'capofamiglia'
            };

            spesa.setId(100);
            dbStub.onFirstCall().resolves([{ membro_id: 5 }]); // spesa di altro membro
            dbStub.onSecondCall().resolves({ affectedRows: 1 });

            const result = await spesa.modificaComeCapofamiglia(capofamiglia.id);

            expect(result).to.be.true;
        });
    });
```