

## Collaudo Elettrodomestici

```
describe('TestElettrodomestici', () => {
  let elettrodomestico;
  let dbStub;

  beforeEach(() => {
    dbStub = sinon.stub(db, 'query');

    elettrodomestico = new Elettrodomestico({
      tipologia: 'Frigorifero',
      nome: 'Samsung RB38',
      marca: 'Samsung',
      modello: 'RB38T605',
      classe_energetica: 'A',
      anno_produzione: 2023,
      anno_acquisto: 2024,
      prezzo: 899.99,
      durata_garanzia: 24,
      abitazione_id: 1
    });
  });

  afterEach(() => {
    dbStub.restore();
  });

  describe('testCalcoloConsumi', () => {
    it('dovrebbe calcolare consumi stimati per classe energetica', () => {
      const consumiStimati = elettrodomestico.calcolaConsumiStimati();

      // Classe A: ~250 kWh/anno per frigorifero
      expect(consumiStimati.annuale).to.be.closeTo(250, 50);
      expect(consumiStimati.mensile).to.be.closeTo(20.8, 5);
    });

    it('dovrebbe calcolare costi mensili stimati', async () => {
      // Mock contratto energia attivo
      dbStub.resolves([
        {
          tipo_utilizzo: 'energia',
          piano_tariffario: 0.25 // €/kWh
        }
      ]);

      const costiStimati = await elettrodomestico.calcolaCostiMensiliStimati();

      expect(costiStimati).to.be.closeTo(5.2, 1); // ~20.8 kWh * 0.25 €/kWh
    });

    it('dovrebbe restituire null senza contratto energia', async () => {
      dbStub.resolves([]);

      const costiStimati = await elettrodomestico.calcolaCostiMensiliStimati();

      expect(costiStimati).to.be.null;
      expect(elettrodomestico.getAvviso()).to.equal('Nessun contratto energia attivo');
    });
  });

  describe('testGaranzia', () => {
    it('dovrebbe verificare stato garanzia', () => {
      const stato = elettrodomestico.verificaGaranzia();

      expect(stato.valida).to.be.true;
      expect(stato.mesiRimanenti).to.be.greaterThan(0);
      expect(stato.scadenza).to.be.instanceOf(Date);
    });

    it('dovrebbe rilevare garanzia scaduta', () => {
      elettrodomestico.setAnnoAcquisto(2022);
      elettrodomestico.setDurataGaranzia(24);

      const stato = elettrodomestico.verificaGaranzia();

      expect(stato.valida).to.be.false;
      expect(stato.mesiRimanenti).to.equal(0);
    });
  });

  describe('testMixEnergetico', () => {
    it('dovrebbe calcolare mix energetico abitazione', async () => {
      dbStub.resolves([
        { nome: 'Frigorifero', consumo_stimato: 250 },
        { nome: 'Lavatrice', consumo_stimato: 200 },
        { nome: 'TV', consumo_stimato: 150 }
      ]);

      const mix = await Elettrodomestico.calcolaMixEnergetico(1);

      expect(mix.totale).to.equal(600);
      expect(mix.percentuali['Frigorifero']).to.be.closeTo(41.7, 1);
      expect(mix.percentuali['Lavatrice']).to.be.closeTo(33.3, 1);
      expect(mix.percentuali['TV']).to.be.closeTo(25, 1);
    });
  });
});
```

# Progettazione

---

## Requisiti non funzionali

Dall'Analisi dei Requisiti emergono i seguenti requisiti non funzionali che impongono vincoli significativi al sistema:

**Usabilità** (RN2): L'interfaccia deve risultare semplice e intuitiva, utilizzabile senza competenze tecniche specifiche, con percorsi chiari e coerenti per la gestione dei flussi di cassa familiari e l'integrazione con i fornitori.

**Performance** (RN3): Il sistema deve garantire tempi di risposta rapidi (entro 3 secondi) per le operazioni più comuni di consultazione e registrazione, considerando la natura web dell'applicazione e l'elaborazione di dati finanziari complessi.

**Disponibilità** (RN4): La piattaforma deve essere disponibile almeno per il 99% del tempo su base mensile, escludendo manutenzioni programmate, per garantire accesso continuo ai dati familiari critici.

**Sicurezza** (RN6, RN8, RN9, RN10, RN11): Il trattamento dei dati personali, economici e contrattuali deve rispettare il GDPR, garantendo protezione contro accessi non autorizzati durante trasmissione e conservazione, con autenticazione robusta e controlli di accesso basati sui ruoli.

**Estendibilità** (RN5): L'architettura deve agevolare l'aggiunta di nuove funzionalità (come OCR degli scontrini, auto-categorizzazione con ML, analisi energetica avanzata) senza compromettere quelle esistenti.

I requisiti di Usabilità e Sicurezza presentano alcuni trade-off, particolarmente nella gestione delle sessioni utente, dove è necessario bilanciare la comodità d'uso con la protezione delle informazioni sensibili. La natura web dell'applicazione richiede particolare attenzione alla sicurezza delle comunicazioni e alla gestione delle sessioni.

## Scelta dell'architettura

Analizzando i vincoli trovati in fase di Analisi del Problema, è stato scelto di utilizzare una architettura Client/Server a 3 livelli distribuiti e indipendenti.

**Client:** è responsabile della presentazione dei dati e dell'interazione con l'utente finale. Comunica con il server tramite una connessione sicura TLS. Esso è progettato per essere multiplatforma e può adattarsi facilmente a contesti d'uso diversi.

**Server:** è stato scelto di dividere le funzionalità di login e log dalla logica di business dell'applicazione, creando quindi 2 server diversi

- ServerGestioneUtenti
- ServerLog

Per molti aspetti anche analizzati sopra e con l'intenzione di mantenere il sistema il più espandibile possibile, si è scelto di sfruttare un'architettura RESTFUL per la comunicazione tra i vari livelli, usando come formato di comunicazione il Json. Si andrebbero così a rispettare, con costi implementativi relativamente bassi, alcuni dei requisiti non funzionali più rilevanti analizzati precedentemente.

# Scelte tecnologiche

## Frontend

Il Client Frontend sarà implementato come **Single Page Application** (SPA) utilizzando un **framework JavaScript** moderno, come **React**, per garantire un'esperienza utente fluida nella gestione dei dati familiari.

Per la comunicazione utilizzeremo **HTTPS obbligatorio** per proteggere i dati finanziari sensibili durante il trasferimento. L'adozione di un client SPA **ottimizza le performance** attraverso:

- **Rendering lato client** che riduce il carico sul server durante la consultazione di report complessi
- **Caching avanzato** delle risorse statiche e dei dati di configurazione famiglia
- **Lazy loading** delle sezioni (report, offerte fornitori) per ottimizzare i tempi di caricamento iniziale

Sviluppi futuri potrebbero includere **Progressive Web App (PWA)** per supporto offline e mobile.

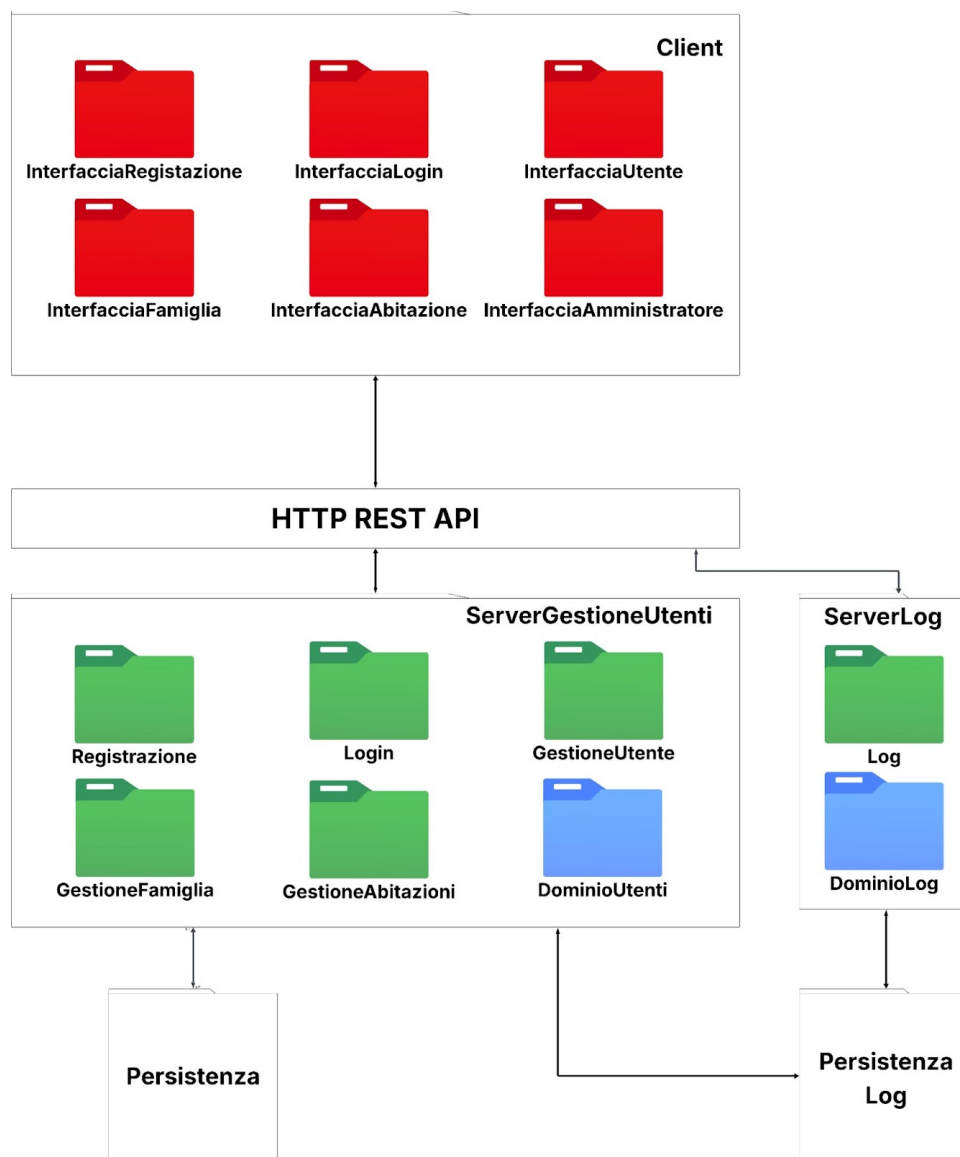
## Backend

Il backend rappresenta il **livello di controllo** e di **logica applicativa** dell'architettura. Ha il compito di gestire le richieste provenienti dai client, elaborare i dati e interagire con il sistema di memorizzazione. Questo livello viene a sua volta articolato in **due componenti**: il **controller**, che racchiude la logica di business dell'applicazione, e la **persistenza**, dedicata alla conservazione dei dati.

Per la gestione dei dati relativi agli utenti e al sistema verrà utilizzato un **DBMS relazionale**: la scelta ricade su **MySQL**, che si dimostra una soluzione affidabile e costantemente aggiornata, anche grazie all'ampia base di utilizzatori.

Infine, viene applicato il **pattern** di progettazione **dependency-inversion** per rilassare i collegamenti tra i componenti del backend, migliorando la futura estensibilità e manutenibilità dell'applicazione.

## Diagramma dei package



## Diagramma dei componenti

