



RISTAT

Progettazione

Membri del team di sviluppo:

Alessandro Musarella
Giovanni Ghirardini
Martino Tommasini

SOMMARIO:

PROGETTAZIONE ARCHITETTURALE	4
REQUISITI NON FUNZIONALI	4
SCELTA DELL'ARCHITETTURA	5
PATTERN UTILIZZATO	7
DIAGRAMMA DEI COMPONENTI	8
PROGETTAZIONE DI DETTAGLIO	9
STRUTTURA	9
INTERAZIONE	24
PROGETTAZIONE DELLA PERSISTENZA	29
PERSISTENZA DATI DI VENDITA	29
PERSISTENZA UTENTI E RISTORANTI	30
PERSISTENZA LOG	30
PROGETTAZIONE DEL COLLAUDO	31
DEPLOYMENT	33
ARTEFATTI	33
DEPLOYMENT TYPE-LEVEL	34

PROGETTAZIONE ARCHITETTURALE

REQUISITI NON FUNZIONALI

Dall'analisi del problema sono emersi i seguenti requisiti non funzionali che impongono dei vincoli al sistema:

- Manutenibilità
- Tempo di risposta
- Usabilità
- Sicurezza
- Disponibilità dei dati

La manutenibilità e modularità del software rappresentano un requisito di particolare importanza nel nostro sistema. Come, infatti, richiesto dal cliente, è necessario avere la possibilità di aggiungere nuove statistiche senza che vengano apportati cambiamenti a statistiche e funzionalità già presenti. Una attenta progettazione deve quindi essere volta ad una astrazione dei modelli di dati, in modo da garantire tale requisito.

Il sistema deve considerare la possibile presenza di una rilevante quantità di informazioni nel tempo costituiti dai dati di vendita provenienti dai ristoranti. Al fine di garantire un tempo di risposta adeguato, il sistema deve assicurare un accesso veloce ai dati memorizzati.

Poiché tale requisito può essere in conflitto con le politiche di sicurezza, è necessario trovare un equilibrio tra i due, considerando di dover garantire un tempo di risposta massimo di 6/7 secondi.

Nel nostro specifico caso, Usabilità e Sicurezza hanno pochi conflitti a parte l'eventuale richiesta di un ulteriore login se per caso scade la sessione di lavoro. Poiché il software basa la sua funzionalità principale sulla possibilità di elaborare risultati statistici sui dati memorizzati, la disponibilità di essi assume un ruolo fondamentale. Poiché non è necessario lavorare su dati real-time è sufficiente predisporre repliche settimanali/bisettimanali dei dati di vendita memorizzati, evitando così eccessivi cali di prestazione del sistema.

SCELTA DELL'ARCHITETTURA

L'architettura considerata più appropriata per questo tipo di sistema è l'architettura client-server a 3 livelli.

L1-Client

Al fine di garantire indipendenza semantica e protezione, ed attenendoci al principio di "minimo privilegio", abbiamo deciso di sviluppare i seguenti client:

- Un client per l'Analista
- Un client per l'Amministratore
- Un client per l'AddettoSicurezza
- Un client per il Ristorante

L2-Server

Abbiamo ritenuto opportuno distinguere i server in base alle funzionalità che offrono:

- Un server per le funzionalità legate all'Analista
- Un server per le funzionalità di gestione legate all'Amministratore
- Un server per le funzionalità legate all'AddettoSicurezza
- Un server per le funzionalità legate al Ristorante
- Un server per le funzionalità legate all'autenticazione

L3-Persistenza

Per avere un accesso ai dati veloce e centralizzato sarà presente un unico server dedicato avente un opportuno DBMS per gestire le strutture dati relative ai dati di vendita. Un ulteriore server sarà dedicato alla replica settimanale/bisettimanale del precedente servizio.

Altri due server sono aggiunti:

- uno per la memorizzazione dei log
- uno per la memorizzazione degli Utenti/Ristoranti registrati nel sistema.

L'utilizzo di server separati garantisce una maggiore resistenza alle compromissioni.

Inoltre, il sistema esterno che fornisce i dati aggiornati di Clienti fidelizzati, ristoranti e cataloghi prodotti/promozioni è collocato su un ulteriore server, su cui è installato DBMS IBM DB2, come stabilito nell'analisi del problema. Per avere una maggiore velocità di accesso ai dati, si è deciso di fare una copia giornaliera delle informazioni relative ai cataloghi, ristoranti e ai clienti fidelizzati all'interno

del nostro server Data persistence. Due sono i vantaggi che derivano da questa soluzione:

- Indipendenza dal DMBS IBM DB2 utilizzato dalla catena di ristoranti
- Tempi di risposta più rapidi dovuti all'eliminazione di query su database posizionati in server diversi

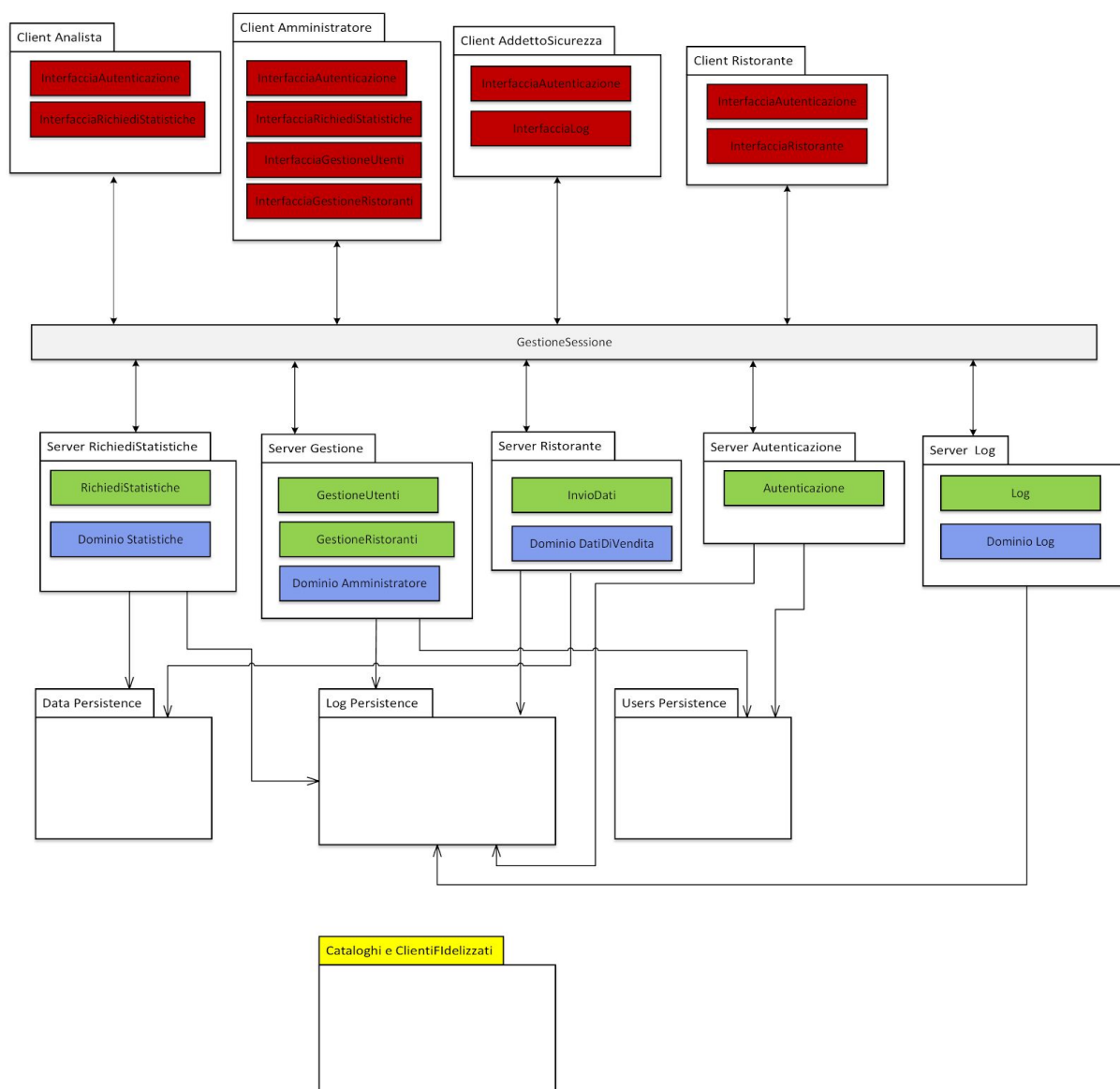
Il meccanismo di backup di tali dati non sarà analizzato all'interno di questa analisi.

Per quanto riguarda le modalità di accesso e scrittura di dati all'interno dei database, si utilizzeranno tecniche basate su forza bruta.

PATTERN UTILIZZATO

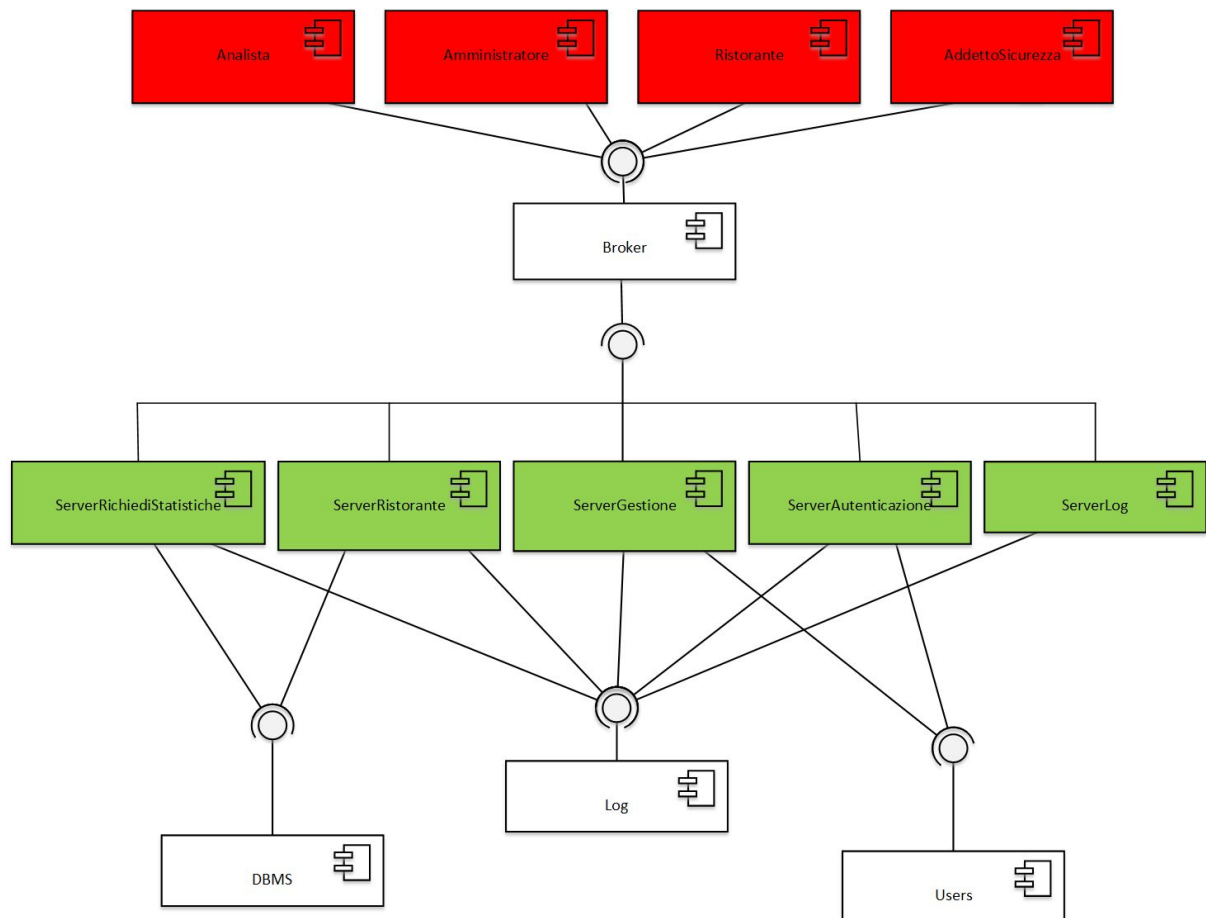
Al fine di avere un maggiore disaccoppiamento tra client e server si è deciso di adottare il pattern Broker. Tale Broker, oltre a gestire la sessione degli utenti e dei ristoranti permette di redirigere le richieste dei client e filtrare le richieste provenienti da client non autorizzati. I server esporranno, quindi i propri servizi, con cui solamente il broker potrà interfacciarsi.

Un canale sicuro sarà implementato con il protocollo TLS tra client e server per garantire la protezione e l'integrità dei dati.



Si osservi che la persistenza "Cataloghi e ClientiFidelizzati" non è direttamente collegata al nostro sistema poiché come già specificato i dati presenti vengono importati periodicamente dai meccanismi di backup all'interno di "Data Persistence".

DIAGRAMMA DEI COMPONENTI



PROGETTAZIONE DI DETTAGLIO

STRUTTURA

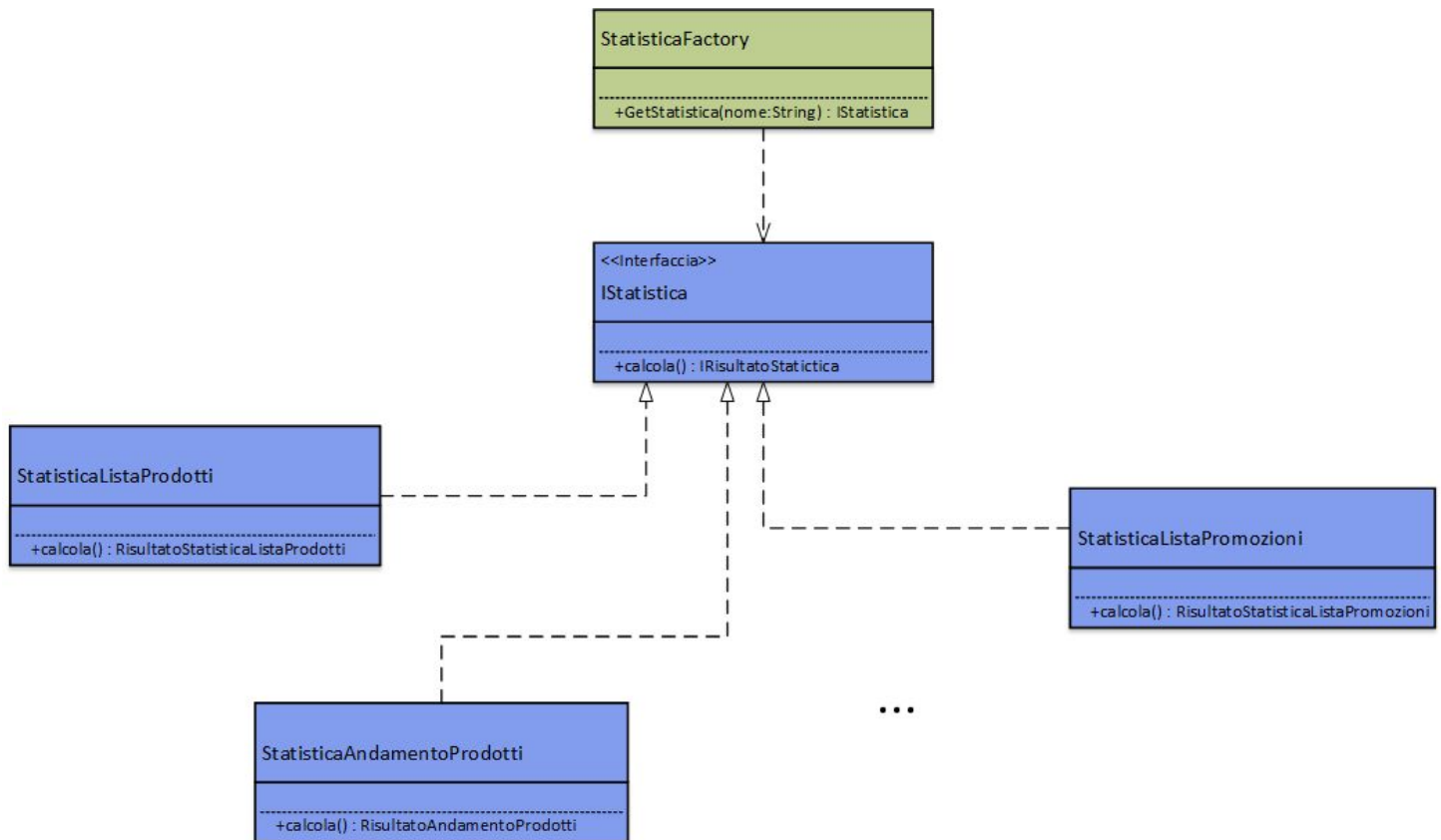
Dato il requisito di dover costruire un sistema che permetta una aggiunta modulare di nuove statistiche, senza apportare modifiche a funzionalità già presenti, la struttura deve essere progettata per tale fine.

A seguire sono riportati i diagrammi di dettaglio delle varie parti del sistema.

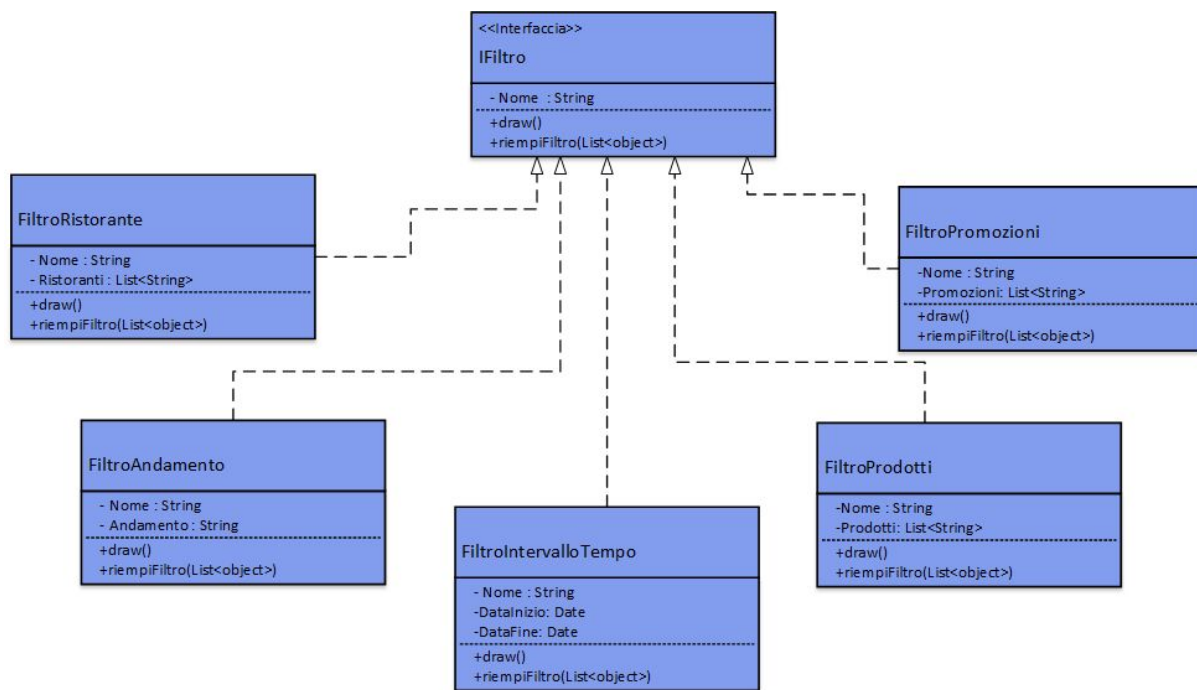
DIAGRAMMA DI DETTAGLIO: Dominio statistiche



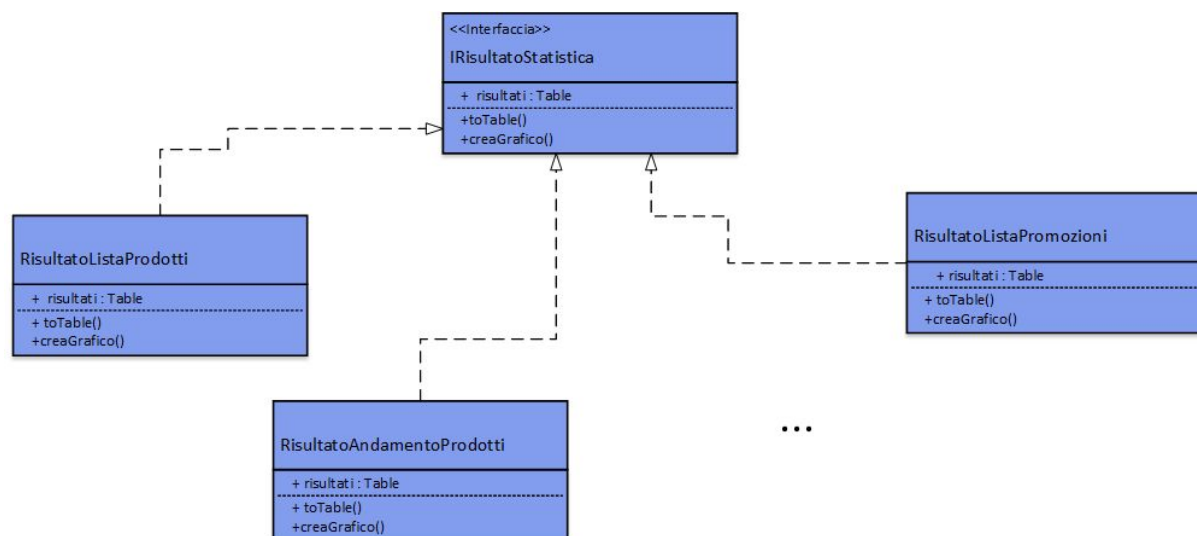
Tali interfacce permettono di nascondere all'utilizzatore le loro effettive implementazioni. Tale scelta permette di avvalersi del pattern Strategy e trarre vantaggio da un comportamento dinamico in base ai tipi che vengono istanziati. Nuove Statistiche, Filtri e RisultatoStatistica possono così essere aggiunti senza portare cambiamenti a codice precedente.



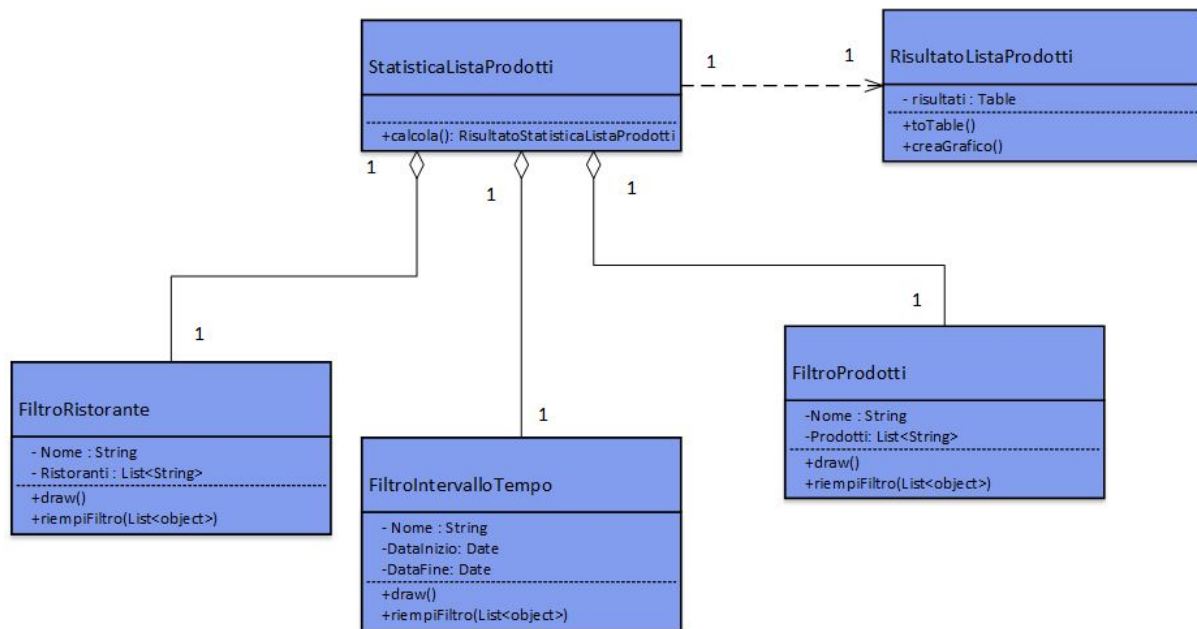
La creazione di una statistica avviene sempre attraverso l'uso di **StatisticaFactory** nascondendo all'utilizzatore la creazione di istanze che implementano l'interfaccia **IStatistica**. Tale interfaccia permette inoltre di realizzare il pattern Strategy: l'algoritmo utilizzato da **IStatistica** per il calcolo della statistica scelta varia in base a quale tipo di statistica è stata precedentemente istanziata. A titolo esemplificativo sono stati riportate solamente 3 realizzazioni dell'interfaccia **IStatistica**. Tutte le statistiche sono riportate nelle pagine a seguire.



Sono qui riportate le diverse realizzazioni della interfaccia IFiltro. Tali tipi concreti implementano i metodi necessari per disegnarsi e popolarsi autonomamente.



Sono qui riportate le diverse realizzazioni della interfaccia I Risultato Statistica. Tali tipi concreti permettono di incapsulare i risultati provenienti dal calcolo della statistica. Poiché i grafici da disegnare variano dinamicamente a seconda dello specifico tipo di risultato statistica, il pattern Strategy permette di modificare dinamicamente l'algoritmo di creazione del grafico e la propria visualizzazione. A titolo esemplificativo sono stati riportate solamente 3 realizzazioni dell'interfaccia I Risultato Statistica. Tutte i risultati sono riportate nelle pagine a seguire.



Il diagramma modella le relazioni di **StatisticaListaProdotti**, una particolare implementazione dell'interfaccia **IStatistica**. Tale classe permette di restituire un'istanza di **RisultatoStatisticaListaProdotti**, utilizzando i filtri di cui la statistica è intrinsecamente composta.

Tale diagramma deve essere preso come esempio per la modellazione di tutte le altre statistiche, ciascuna legate ai Filtri e **RisultatoStatistica** relativi. Nella pagina seguente sono elencate le loro relazioni.

OSSERVAZIONI

Grazie alla strutturazione del modello dominio-statistiche sopra definita, l'aggiunta di una nuova statistica non comporta cambiamento a codice precedentemente scritto ma alla creazione di una nuova classe che realizza l'interfaccia **IStatistica**.

Nel caso la nuova statistica richieda filtri non attualmente esistenti, possono esserne creati nuovi che implementano l'interfaccia **IFiltro**.

Nel caso la nuova statistica richieda un risultato statistica non attualmente esistente, può esserne creato uno nuovo che implementa l'interfaccia **IRisultatoStatistica**.

ELENCO STATISTICHE, FILTRI E RISULTATI

A fronte della varietà delle statistiche proposte dal software è opportuno elencare il nome della statistica, i filtri collegati e il suo risultato per facilitare il team di implementazione.

Legenda:

Nome Statistica	Filtri	Nome Risultato
Descrizione		

StatisticaListaProdotti	FiltroIntervalloTempo, FiltroRistoranti, FiltroProdotti	RisultatoListaProdotti
Restituisce il numero di unità vendute e la percentuale di vendite attraverso promozione di ogni prodotto.		

StatisticaConfrontoRistorantiProdotti	FiltroIntervalloTempo, FiltroRistoranti, FiltroProdotti	RisultatoConfrontoRistorantiProdotti
Restituisce la lista dei prodotti confrontati per numero di vendite tra i ristoranti selezionati.		

StatisticaAndamentoProdotti	FiltroIntervalloTempo, FiltroAndamento, FiltroRistoranti, FiltroProdotti	RisultatoAndamentoProdotti
Restituisce il numero di vendite giornaliere, settimanali o mensili di determinati prodotti.		

StatisticaListaPromozioni	FiltroIntervalloTempo, FiltroRistoranti, FiltroPromozioni	RisultatoListaPromozioni
Restituisce il numero di unità vendute di una o più promozioni.		

StatisticaConfrontoRistorantiPromozioni	FiltroIntervalloTempo, FiltroRistoranti, FiltroProdotti	RisultatoConfrontoRistorantiPromozioni
--	---	--

Restituisce la lista delle promozioni confrontate per numero di vendite tra i ristoranti selezionati.

StatisticaAndamentoPromozioni	FiltroIntervalloTempo, FiltroAndamento, FiltroRistoranti, FiltroPromozioni	RisultatoAndamentoPromozioni
--------------------------------------	---	------------------------------

Restituisce il numero di vendite giornaliere, settimanali o mensili di determinate promozioni.

StatisticaEtaClientiProdotti	FiltroIntervalloTempo, FiltroRistoranti	RisultatoEtaClientiProdotti
-------------------------------------	--	-----------------------------

Restituisce il prodotto più venduto per fascia di età.
(0-20)(20-30)(30-40)(40-50)(50-60)(60-70)(70+)

StatisticaSessoClienti	FiltroIntervalloTempo, FiltroRistoranti	RisultatoEtaClientiProdotti
-------------------------------	--	-----------------------------

Restituisce la quantità dei clienti per ogni sesso

StatisticaEtaClienti	FiltroIntervalloTempo, FiltroRistoranti	RisultatoEtaClienti
-----------------------------	--	---------------------

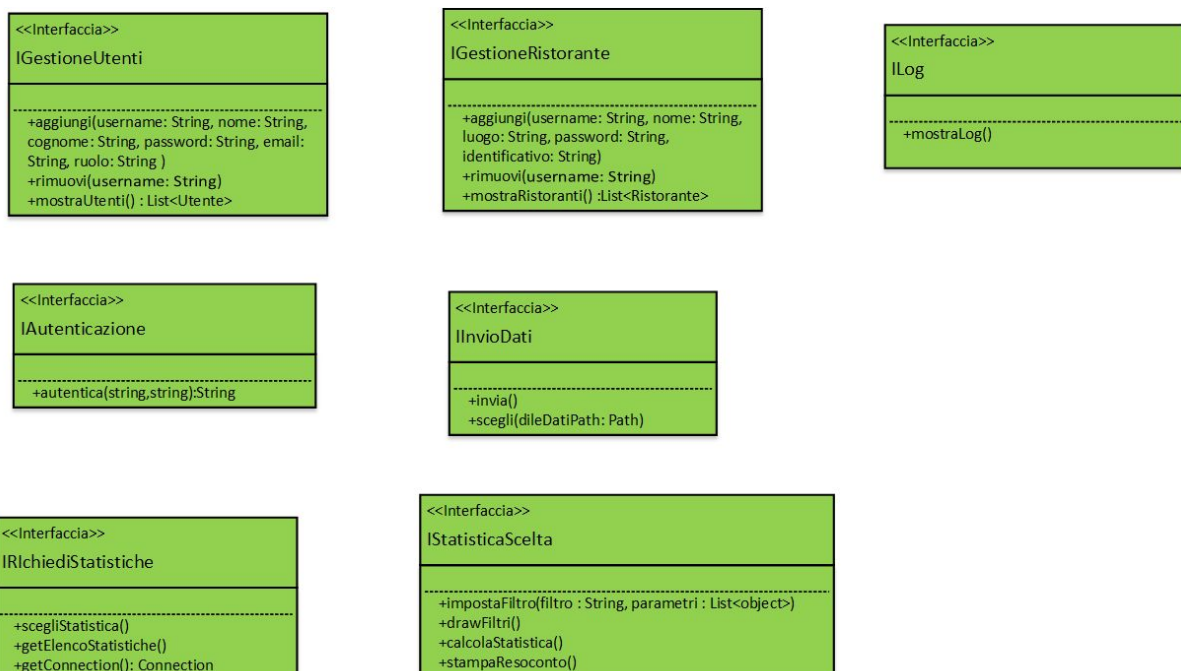
Restituisce il numero di clienti per fascia di età.
(0-20)(20-30)(30-40)(40-50)(50-60)(60+)

StatisticaPercentualeFidelity	FiltroIntervalloTempo, FiltroRistoranti	RisultatoPercentualeFidelity
--------------------------------------	--	------------------------------

Restituisce la percentuale dei clienti fidelizzati sulla totalità dei clienti

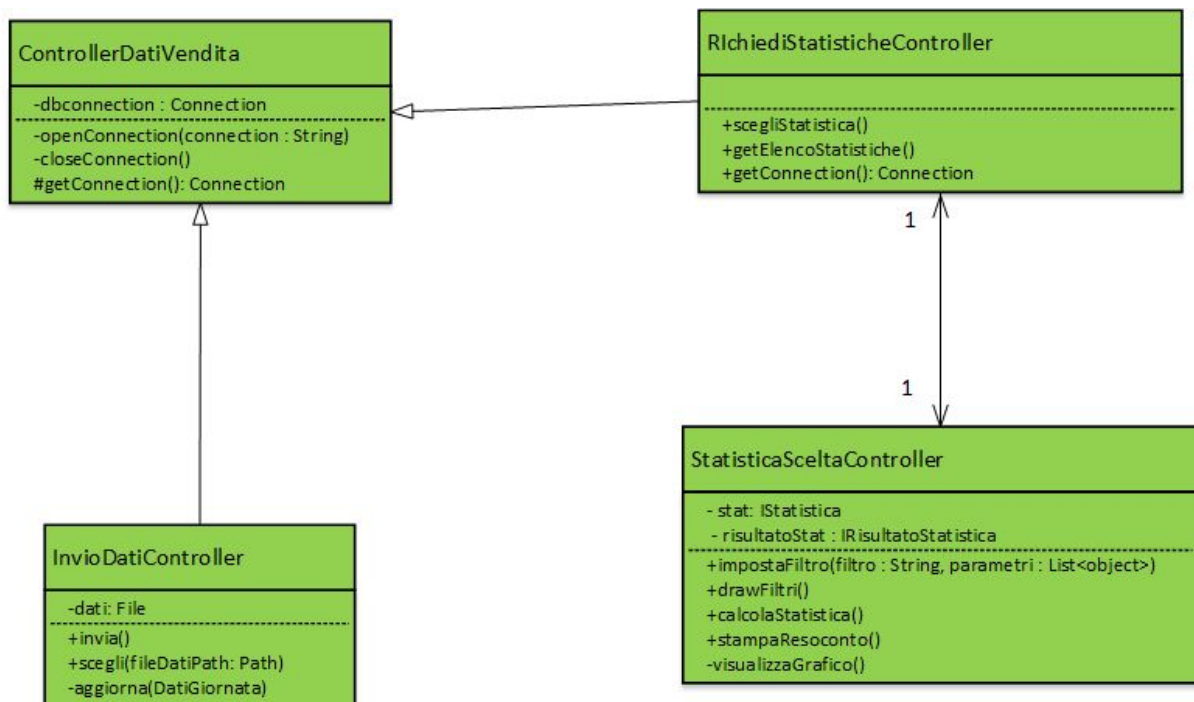
StatisticaPercentualeBusiness	FiltroIntervalloTempo, FiltroRistoranti	RisultatoPercentualeBusiness
Restituisce la percentuale dei clienti business sulla totalità dei clienti		
StatisticaListaProdottiBusiness	FiltroIntervalloTempo, FiltroRistoranti	RisultatoListaProdottiBusiness
Restituisce una lista ordinata dei prodotti più venduti ai clienti business.		
StatisticaListaPromozioniBusiness	FiltroIntervalloTempo, FiltroRistoranti	RisultatoListaPromozioniBusiness
Restituisce una lista ordinata delle promozioni più vendute ai clienti business.		
StatisticaRapportoBusiness	FiltroIntervalloTempo, FiltroRistoranti	RisultatoRapportoBusiness
Restituisce il rapporto pranzo/cena relative ai clienti business		

DIAGRAMMA DI DETTAGLIO: Interfacce nei server



Attraverso l'uso di interfacce possiamo applicare il Dependency Inversion Principle, in modo da dipendere unicamente da astrazioni.

DIAGRAMMA DI DETTAGLIO: RichiediStatistiche & InvioDati

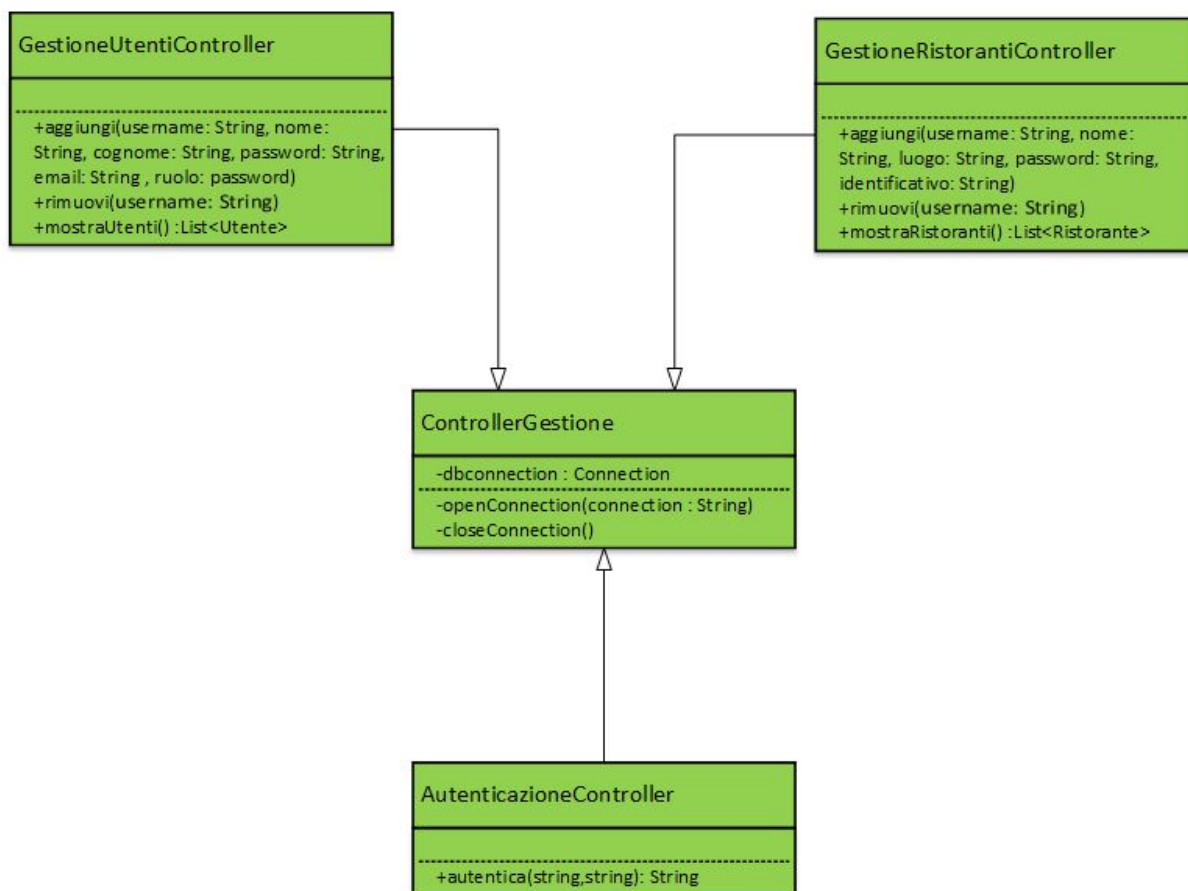


Si è deciso di fattorizzare l'accesso al database dei dati di vendita utilizzando la classe **ControllerDatiVendita**, estesa dai controller. In questo modo i controller avranno a disposizione i metodi per collegarsi al database, ereditandoli dal padre. La classe **StatisticaSceltaController** potrà accedere alla persistenza grazie alla associazione bidirezionale con **RichiediStatisticheController**.

L'**InvioDatiController** si occupa di gestire i dati che vengono inviati dal ristorante. In particolare, converte il contenuto del file JSON caricato dal Ristorante in una istanza della classe **DatiGiornata**, scrivendo successivamente nel database. I dati sarebbero potuti essere inseriti direttamente in database, senza la creazione di una istanza di **DatiVendita**. Tuttavia, tale scelta permette di scrivere sul database in modo indipendente dal tipo di file che viene utilizzato, a patto di convertire il suo contenuto in **DatiVendita**.

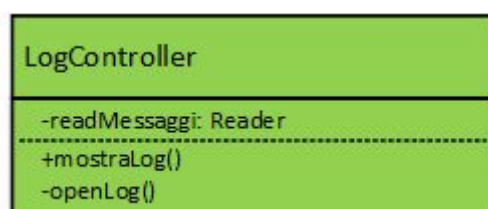
Le istanze presenti all'interno di **StatisticaSceltaController** sono istanziate dinamicamente da **StatisticaFactory**, fornendo così un comportamento dinamico in base al tipo dell'istanza.

DIAGRAMMA DI DETTAGLIO: GestioneUtenti & GestioneRistoranti & Autenticazione



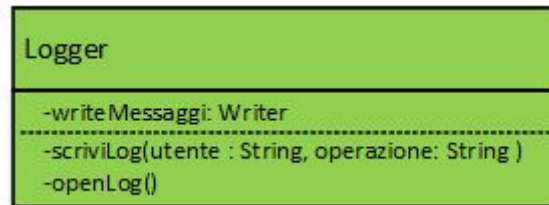
Come in precedenza, si è deciso di fattorizzare l'accesso alla persistenza degli utenti utilizzando la classe `ControllerGestione` che viene estesa dai controller che accedono a tale database.

DIAGRAMMA DI DETTAGLIO: Log



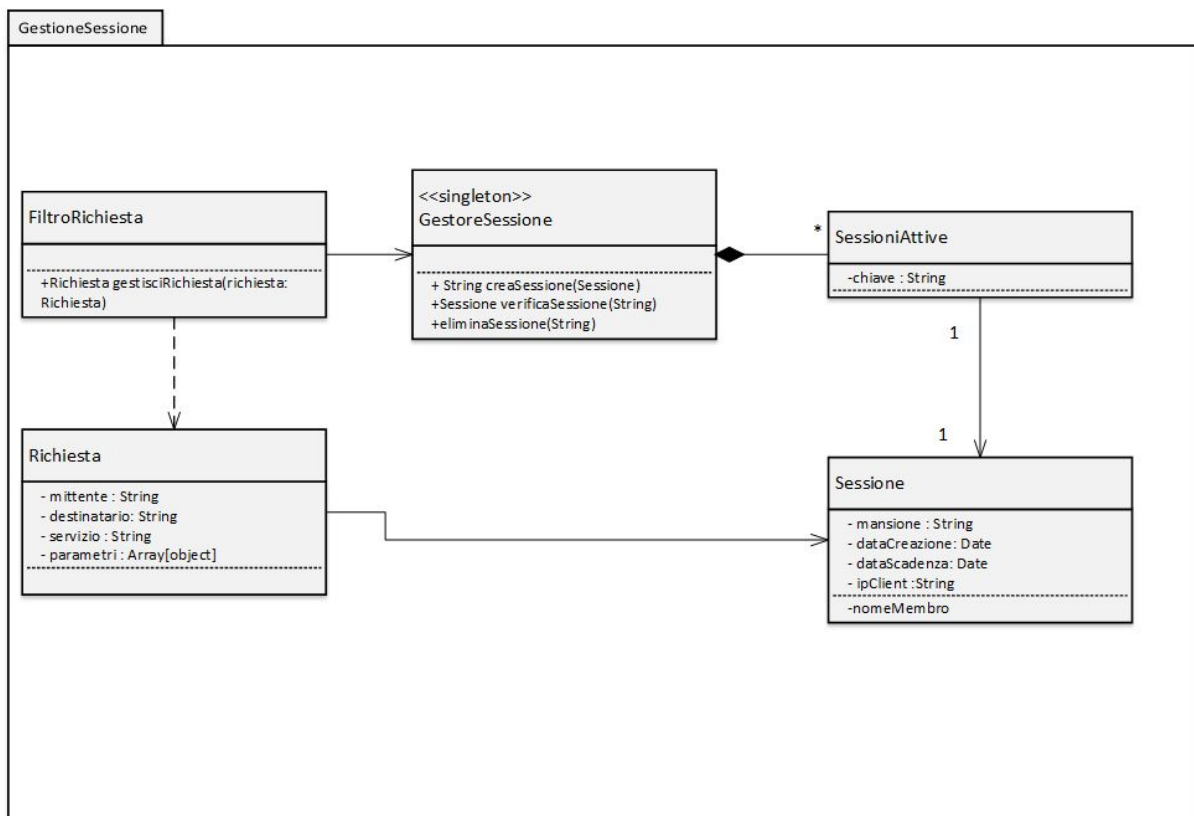
Tale controller si occupa di leggere dai file di log per mostrarli all'AddettoSicurezza. Si è deciso di non gestire la scrittura dei log all'interno di questa entità per non appesantire il traffico che circola nel broker.

La soluzione adottata è l'introduzione di una classe Logger:



Tale classe viene ereditata da `ControllerDatiVendita` e `ControllerGestione` per permettere alle classi figlie la memorizzazioni delle operazioni sui log.

BROKER



Viene qui descritta la struttura del Broker. La classe **FiltroRichiesta** si occupa di gestire le richieste che arrivano dai vari client.

Le richieste vengono inoltrate al server corrispondente.

Le risposte vengono inoltrate al client di destinazione.

DIAGRAMMA DI DETTAGLIO: Home RichiediStatistica & View StatisticaScelta



La Home permettere di scegliere la statistica desiderata tra quelle disponibili, divise per aree (Statistiche Prodotti, Statistiche Promozioni, Statistiche Clienti Fidelizzati, Statistiche Business).

La View StatisticaScelta è unica per tutte le statistiche ma viene popolata dinamicamente in base ai filtri che compongono ogni statistica.

I filtri vengono mostrati con un valore di default che varia in base al tipo di filtro.

L'Analista può applicare i filtri e richiedere il calcolo della statistica premendo il bottone calcola.

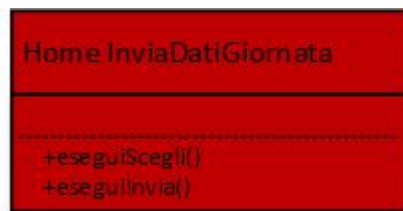
Il risultato della statistica è mostrato nella parte a sinistra della view StatisticaScelta.

Il grafico, se possibile per tale risultato, viene mostrato sulla sua destra.

Attraverso il pulsante StampaResoconto, l'Analista può scaricare i risultati per salvarli nel proprio filesystem.

The screenshot shows the RISTAT web application interface. On the left is a blue sidebar with the RISTAT logo and a menu containing: **- Statistiche Prodotti** (with sub-items: Lista Prodotti, Confronto Ristoranti, **Andamento Vendite**), **+ Statistiche Promozioni**, **+ Statistiche Clienti Fidelizzati**, and **+ Statistiche Clienti Business**. The main content area has a light blue header with three dropdown menus: **Prodotto:**, **Ristoranti:**, and **Andamento:**, followed by a **Calcola** button. Below the header, the main area is split into two columns. The left column is labeled **Esempio di risultato** and the right column is labeled **Esempio di grafico**. At the bottom right of the right column is a **StampaResoconto** button.

DIAGRAMMA DI DETTAGLIO: Home InviaDatiGiornata



La Home InviaDatiGiornata è la schermata utilizzata dal Ristorante per inviare il file JSON contenente i dati di vendita.

Il Ristorante può scegliere il file da caricare (solamente file JSON saranno accettati validando l'input).

Attraverso il bottone d'invio il Ristorante conferma l'operazione.

The screenshot shows the 'RISTAT' application interface. At the top, there is a blue header bar with the 'RISTAT' logo on the left and the title 'Invio Dati Giornata' on the right. Below the header, the main content area is white. It features a large, empty gray rectangular box for file selection. To the right of this box is a blue button labeled 'Scegli'. Below the gray box, centered, is another blue button labeled 'Invia'.

DIAGRAMMA DI DETTAGLIO: Home GestioneRistoranti



Il diagramma sopra rappresentato mostra le funzionalità di gestione dei ristoranti da parte dell'amministratore.

RISTAT	Gestione Ristoranti			
	Username	IdRistorante	Nome	Luogo
	Rist01	4145	SuperRistor	Brodano(MO)

Left sidebar menu:

- + Statistiche Prodotti
- + Statistiche Promozioni
- + Statistiche Clienti Fidelizzati
- + Statistiche Clienti Business
- Gestione Utenti
- Gestione Ristoranti

Aggiungi Ristorante

Form fields:

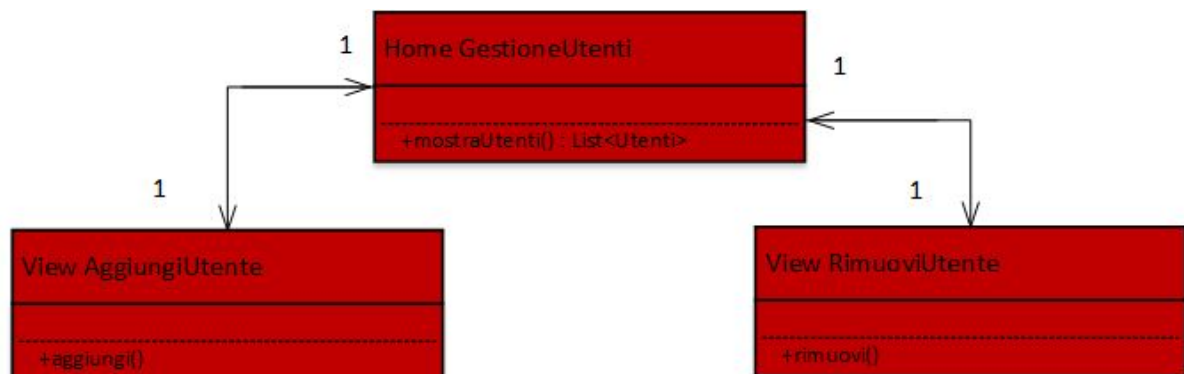
- Username
- IdRistorante
- Nome
- Luogo
- Password
- Ripeti Password

Buttons: Annulla, Conferma

Rimuovere Rist01?

Buttons: Annulla, Conferma

DIAGRAMMA DI DETTAGLIO: Home GestioneUtenti



Il diagramma sopra rappresentato mostra le funzionalità di gestione degli utenti da parte dell'amministratore.

RISTAT

+ Statistiche Prodotti
+ Statistiche Promozioni
+ Statistiche Clienti Fidelizzati
+ Statistiche Clienti Business

Gestione Utenti

Gestione Ristoranti

Username


Nome

Cognome

Email

Ruolo

Lupo00	Lupo	Lucio	l.lucio@mele.com	AN	
				AM	
				AD	



Aggiungi Utente

Username

Nome

Cognome

Email

Ruolo

Password

Ripeti Password

Annulla

Conferma

Rimuovere Lupo01?

Annulla

Conferma

DIAGRAMMA DI DETTAGLIO: Home VisualizzaLog



La Home VisualizzaLog visualizza i log del sistema. Ogni entry è composta dalla data, dall'username dell'utente e dal messaggio del log.

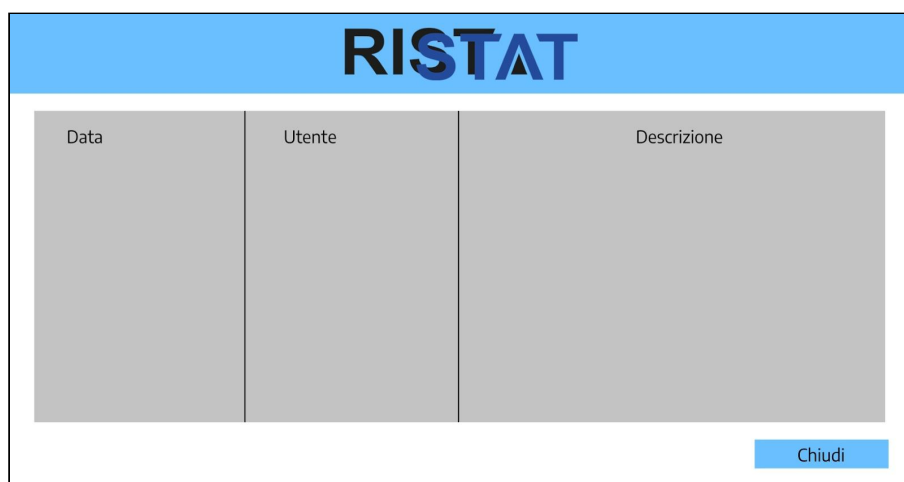
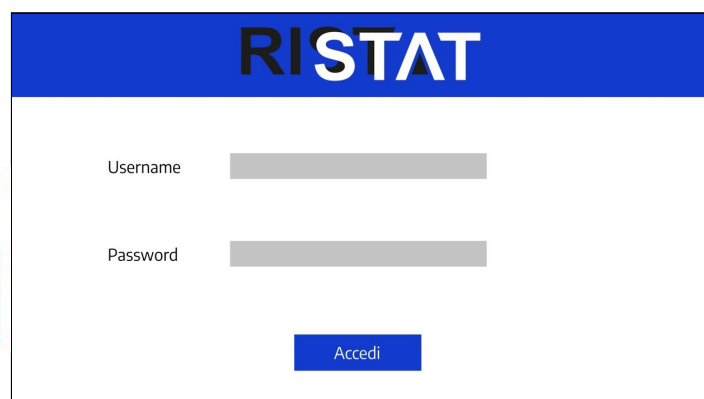
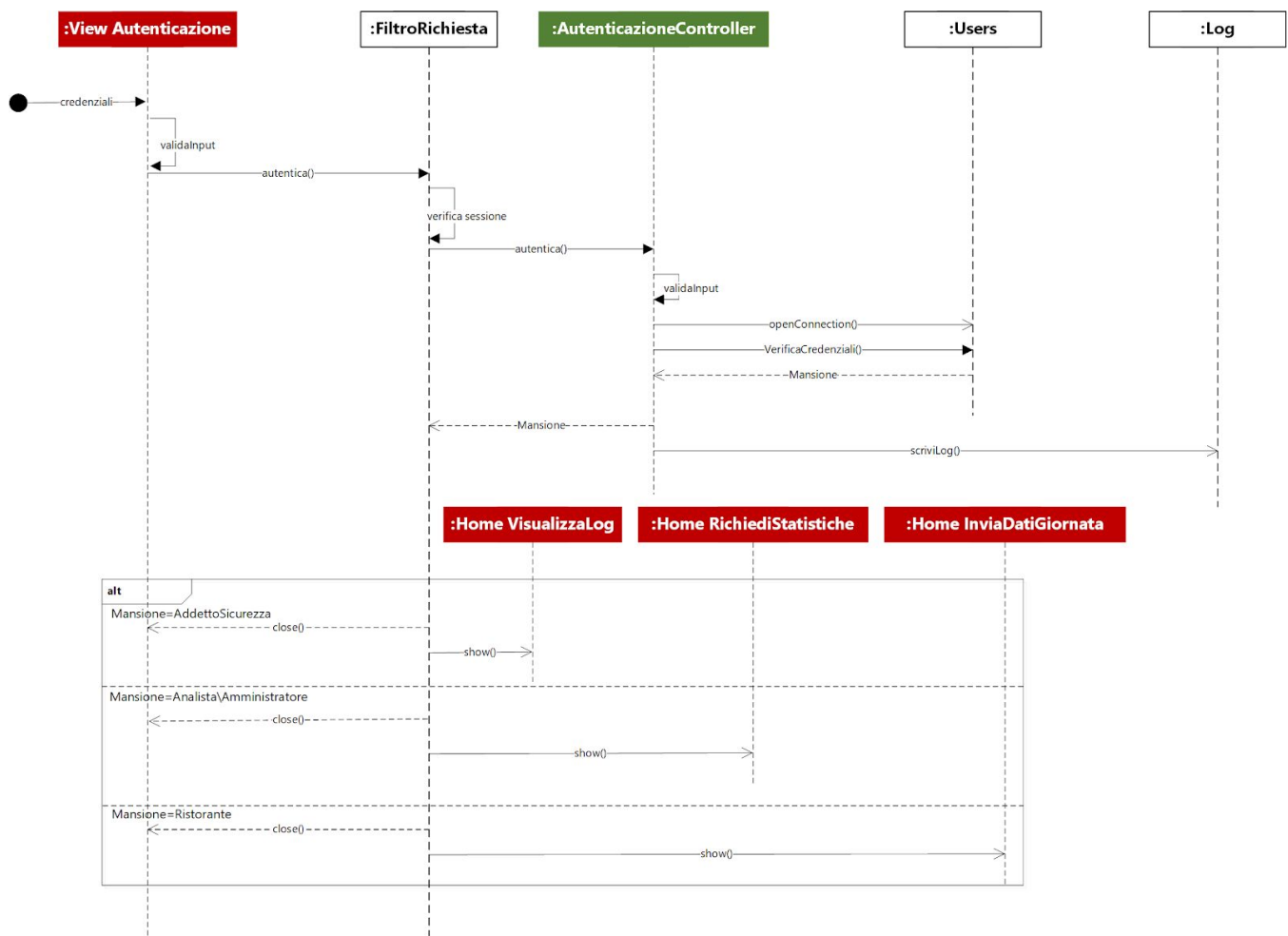


DIAGRAMMA DI DETTAGLIO: Autenticazione



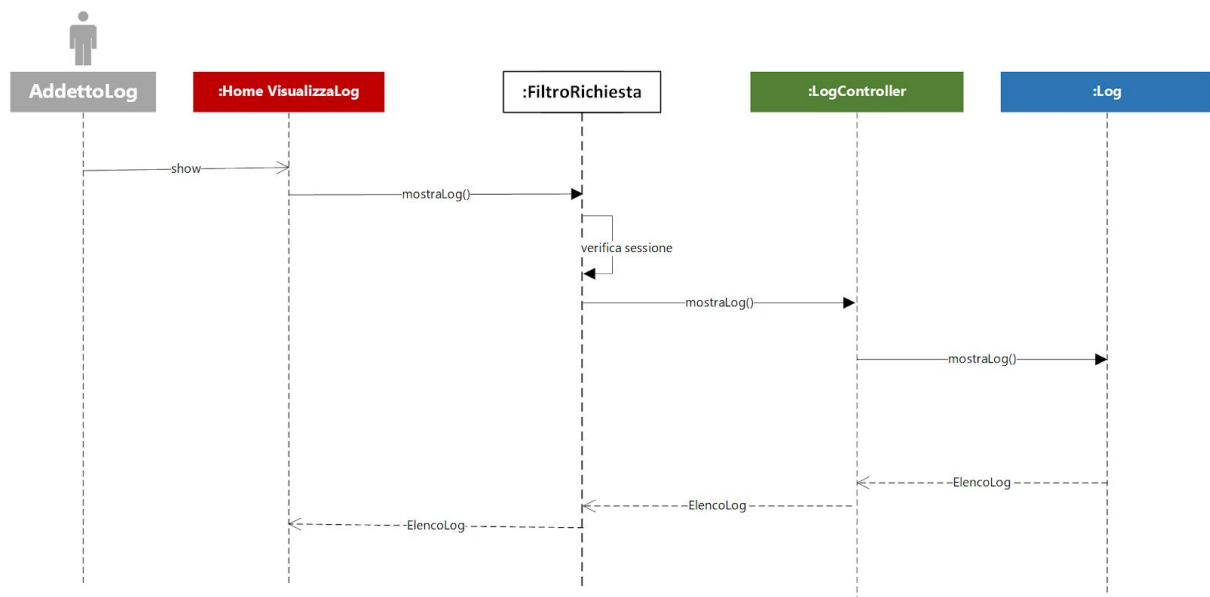
INTERAZIONE

DIAGRAMMA DI SEQUENZA: Autenticazione



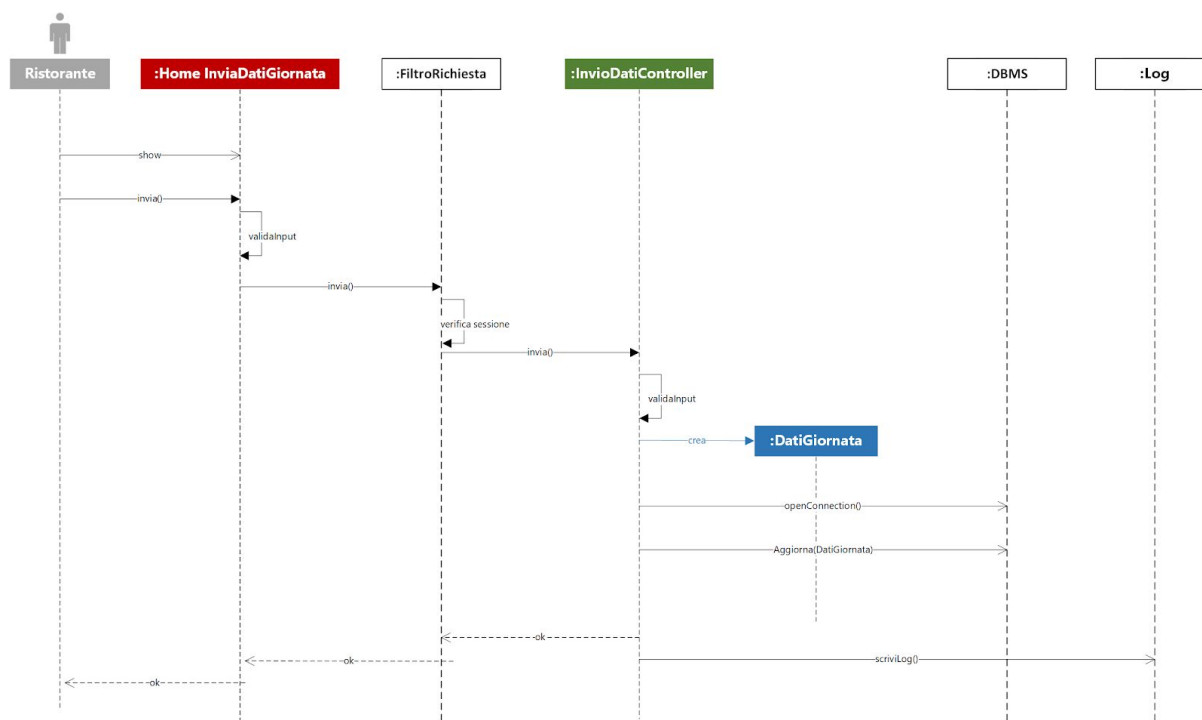
Il diagramma mostra il protocollo per l'autenticazione degli Utenti e dei Ristoranti, basato sulla restituzione di una Mansione che permetterà la corretta visualizzazione da parte di chi accede. Sono necessarie le validazione degli input sia da parte client che server così da controllare che non vi siano state variazioni durante la comunicazione.

DIAGRAMMA SEQUENZA: Log



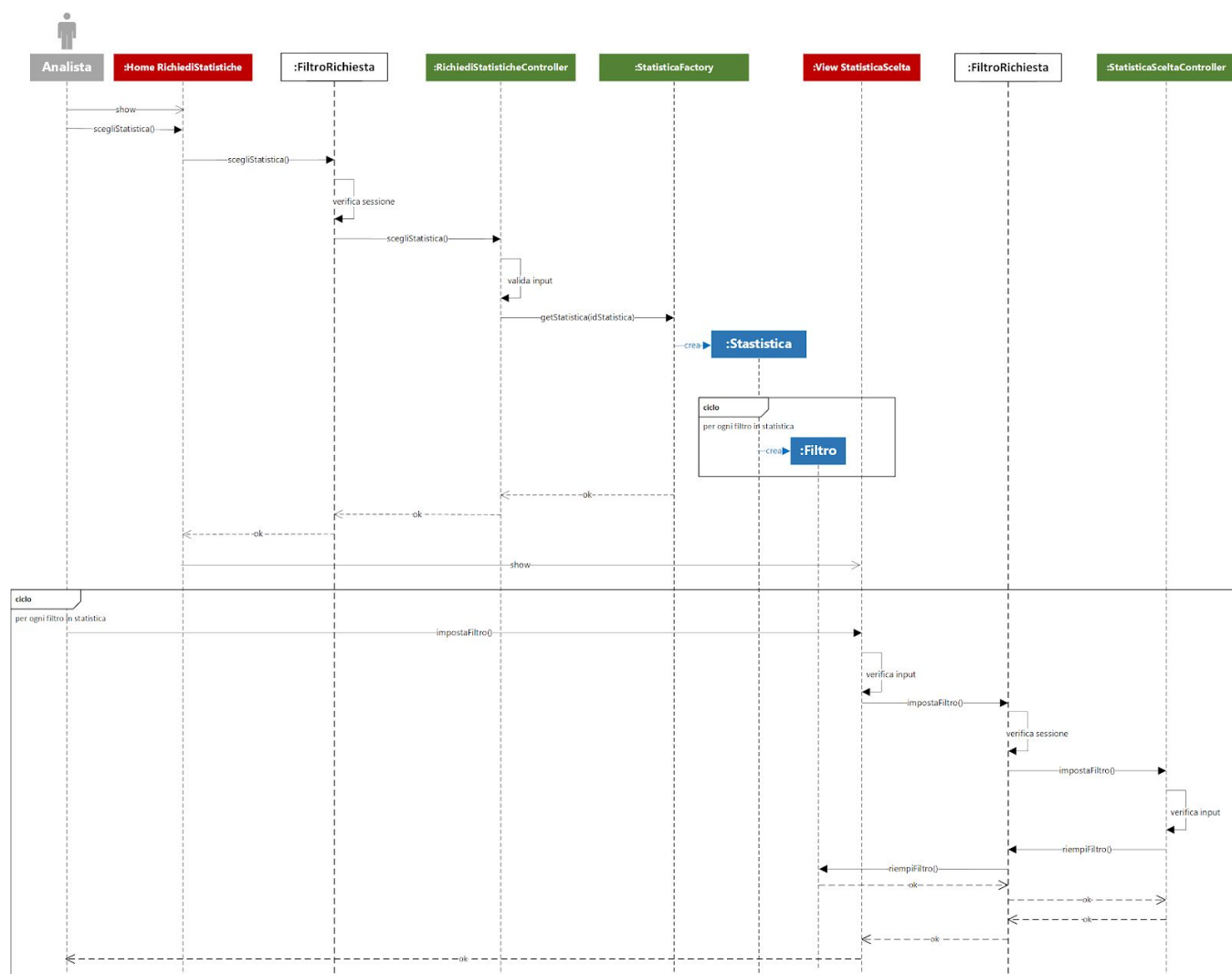
Anche se non specificato la lettura dei log avviene accadendo alla persistenza dei log

DIAGRAMMA DI SEQUENZA: Invio Dati



Il diagramma mostra l'invio dei dati di giornata da parte di un Ristorante. Sono necessarie le validazione degli input sia da parte client che server così da controllare che non vi siano state variazioni durante la comunicazione. Il contenuto del file JSON non viene direttamente memorizzato nel database ma ciò avviene attraverso la creazione di una istanza di **DatiGiornata**. Il vantaggio di tale approccio è stato precedentemente spiegato nei diagrammi di **InvioDatiController**.

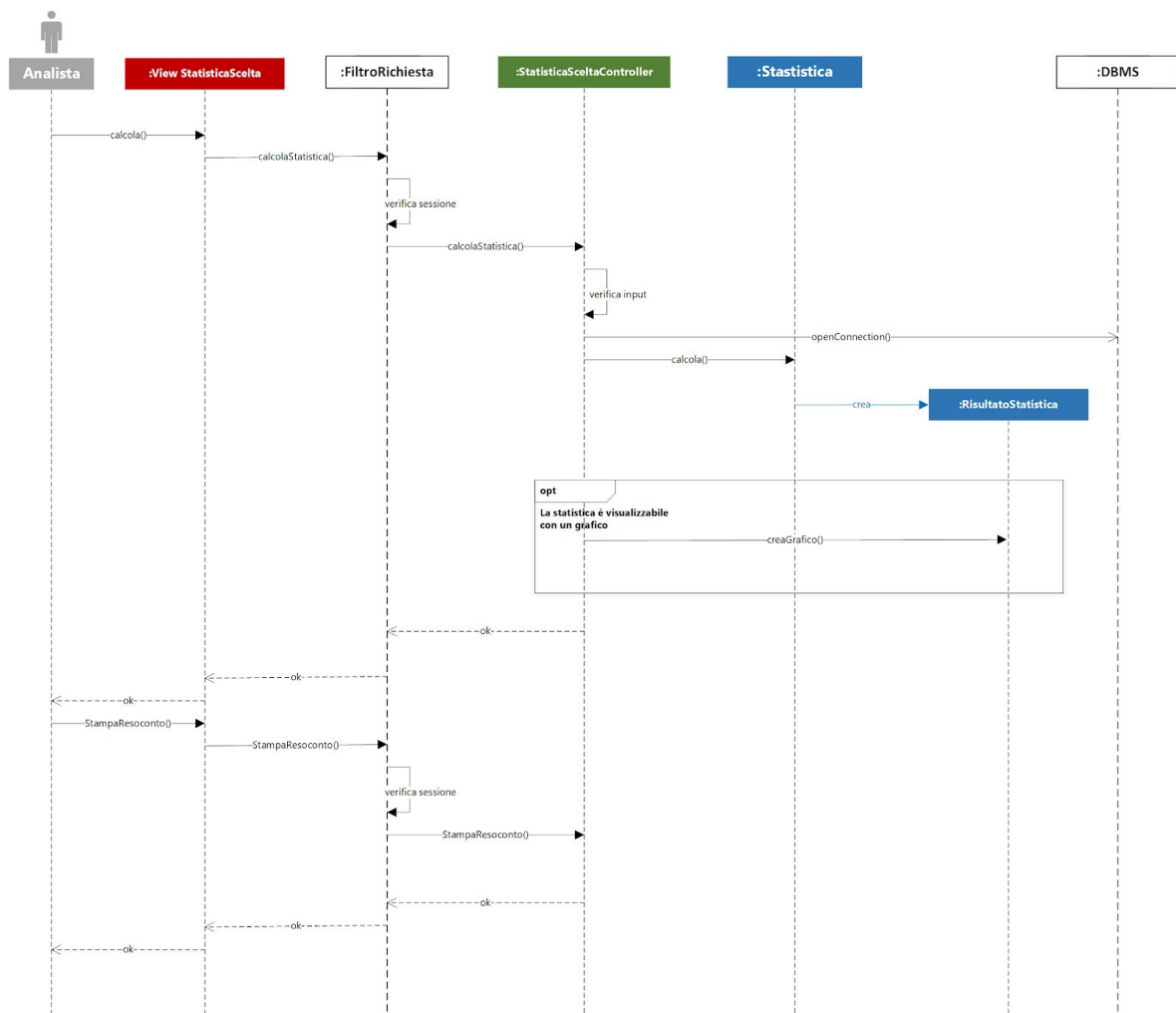
DIAGRAMMA DI SEQUENZA: Richiedi Statistica



Il diagramma descrive la selezione della statistica e dei rispettivi filtri. Alla scelta della statistica viene istanziato il tipo statistica opportuna attraverso la Factory e i tipi di filtri corrispondenti.

La doppia presenza della classe FiltroRichiesta del broker è stata inserita per facilitare la lettura del grafico.

DIAGRAMMA SEQUENZA: Calcola Statistica

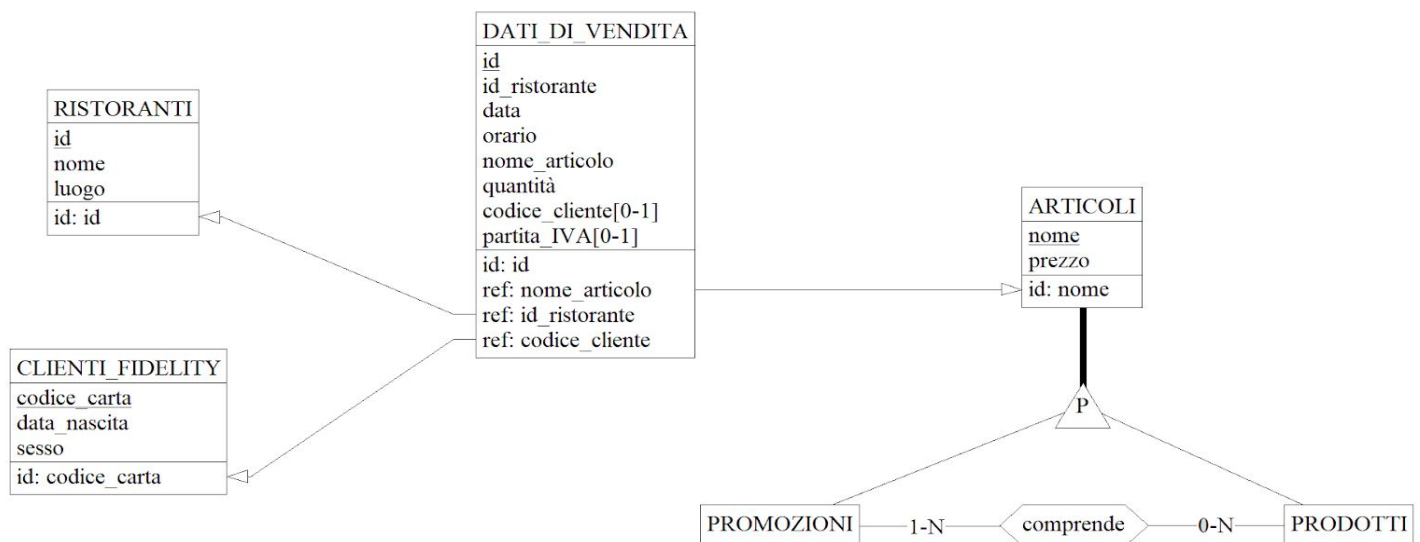


Il diagramma è ripreso da quello già analizzato nella analisi del problema. è stata aggiunta l'interazione con il database dei dati di vendita per le query necessarie al funzionamento del programma di statistiche. Tali interrogazioni non sono riportate perché nascoste all'interno del metodo calcola().

PROGETTAZIONE DELLA PERSISTENZA

PERSISTENZA DATI DI VENDITA

Poiché i dati gestiti sono altamente strutturati, si è deciso di utilizzare un database relazionale. La scelta del DBMS è ricaduta su PostgreSQL, in grado di fornire un supporto adatto per le quantità di dati da noi trattate.



Abbiamo ritenuto necessario introdurre un'ulteriore entità ARTICOLI per poter rappresentare il fatto che dati di vendita può contenere sia un singolo prodotto che una promozione.

PERSISTENZA UTENTI E RISTORANTI

UTENTI
<u>username</u>
hash_password
nome
cognome
email
ruolo
id: username

RISTORANTI
<u>username</u>
hash_password
identificativo
nome
luogo
id: username

La password è memorizzata utilizzando un hash per avere maggiore sicurezza.

Il database deve essere messo in sicurezza fisicamente in quanto contenente informazioni sensibili. Inoltre, prevedere politiche di accesso più ristrette.

PERSISTENZA LOG

Si è scelto di utilizzare un file come meccanismo di memorizzazione dei log di sistema. I file di log devono essere opportunamente protetti.

Formato:

“Timestamp ruolo username operazione”

PROGETTAZIONE DEL COLLAUDO

```
[TestFixture]
public class TestStatistica
{
    readonly string STAT_LISTA_PRODOTTI = "lista_prodotti";

    private IStatistica _stat;

    [SetUp]
    public void StatisticaSetUp()
    {
        StatisticaFactory statFactory = new StatisticaFactory();
        _stat = statFactory.getStatistica(STAT_LISTA_PRODOTTI);
    }

    [Test]
    public void TestTypeListaProdotti()
    {
        // controllo se la Factory ha istanziato il giusto tipo
        Assert.IsInstanceOf(typeof(StatisticaListaProdotti), _stat);

        // controllo se la Statistica contiene i filtri corretti
        List<IFiltro> filtri = _stat.getFiltri();
        Assert.That(filtri.Count, Is.EqualTo(3));

        bool trovatoFiltroProdotti = false;
        bool trovatoFiltroIntervalloTempo = false;
        bool trovatoFiltroRistoranti = false;

        foreach(IFiltro filtro in filtri)
        {
            if (filtro.GetType() == typeof(FiltroProdotti))
                trovatoFiltroProdotti = true;
            if (filtro.GetType() == typeof(FiltroIntervalloTempo))
                trovatoFiltroIntervalloTempo = true;
            if (filtro.GetType() == typeof(FiltroRistoranti))
                trovatoFiltroRistoranti = true;
        }

        Assert.IsTrue(trovatoFiltroProdotti);
        Assert.IsTrue(trovatoFiltroIntervalloTempo);
        Assert.IsTrue(trovatoFiltroRistoranti);

        // controllo se la Statistica restituisce il tipo di risultato corretto
        Assert.IsInstanceOf(typeof(RisultatoListaProdotti), _stat.calcola());
    }
}

[TestFixture]
public class TestUtenti
{
    GestioneUtentiController _Ucontroller;

    [SetUp]
    public void UtentiSetUp()
    {
        _Ucontroller = new GestioneUtentiController();
    }

    [Test]
```

```

public void TestGestioneUtenti()
{
    // controllo aggiunta utente
    Utente utente = new Utente("LupoLucio98", "Lupo", "Lucio",
"lupo.lucio@melevisione.com", "Analista");
    _Ucontroller.aggiungi("LupoLucio98", "Lupo", "Lucio",
"passwordSegretissima", "lupo.lucio@melevisione.com", "Analista");
    List<Utente> utenti = _Ucontroller.mostraUtenti();

    Utente utenteAggiunto = null;

    foreach(Utente u in utenti)
    {
        if (utente.getUsername().equals(u.getUsername()))
            utenteAggiunto = u;
    }

    Assert.IsNotNull(utenteAggiunto);

    Assert.AreEqual(utente.getNome(), utenteAggiunto.getNome());
    Assert.AreEqual(utente.getCognome(), utenteAggiunto.getCognome());
    Assert.AreEqual(utente.getEmail(), utenteAggiunto.getEmail());
    Assert.AreEqual(utente.getRuolo(), utenteAggiunto.getRuolo());
}

}

[TestFixture]
public class TestRistoranti
{
    GestioneRistorantiController _Rcontroller;

    [SetUp]
    public void RistorantiSetUp()
    {
        _Rcontroller = new GestioneRistorantiController();
    }

    [Test]
    public void TestGestioneRistoranti()
    {
        // controllo aggiunta ristorante
        Ristorante rist = new Ristorante("toto&peppino", "p123",
"PizzeriaDaTotoEPeppino", "Bologna");
        _Rcontroller.aggiungi("toto&peppino", "passwordMenoSegreta", "p123",
"PizzeriaDaTotoEPeppino", "Bologna");
        List<Ristorante> ristoranti = _Rcontroller.mostraRistoranti();

        Ristorante ristAggiunto = null;

        foreach (Ristorante r in ristoranti)
        {
            if (rist.getUsername().equals(r.getUsername()))
                ristAggiunto = r;
        }

        Assert.IsNotNull(ristAggiunto);

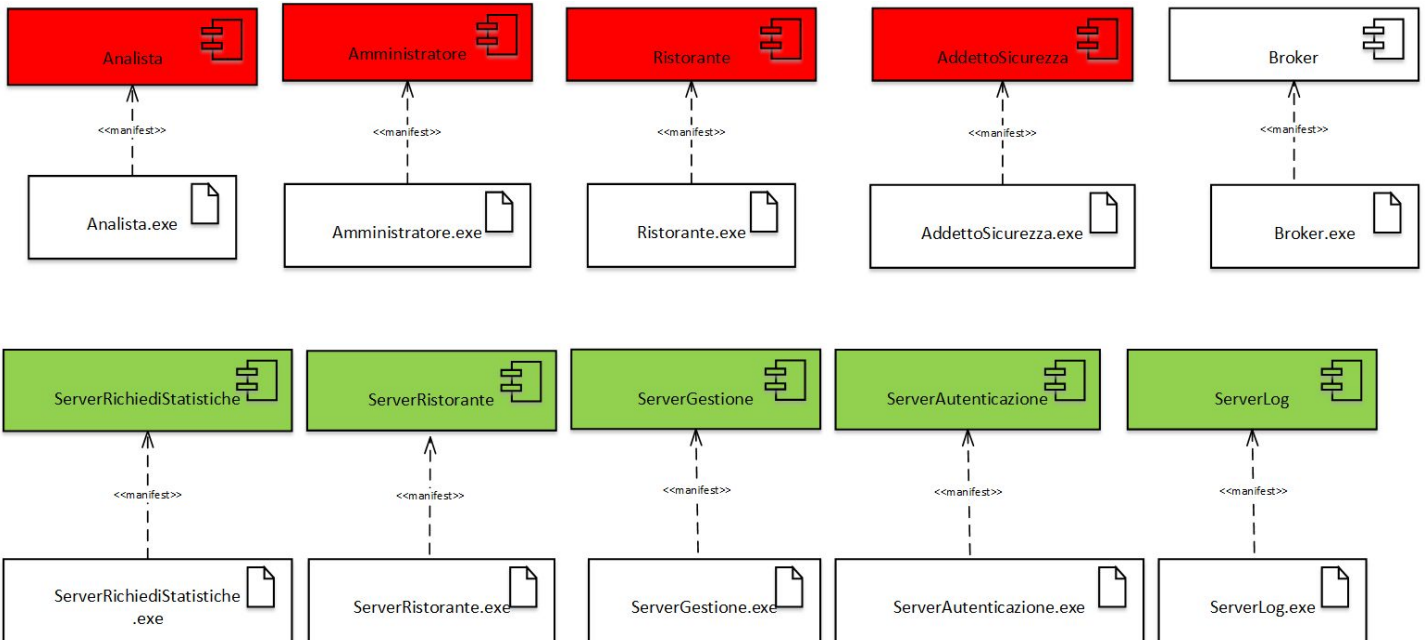
        Assert.AreEqual(rist.getNome(), ristAggiunto.getNome());
        Assert.AreEqual(rist.getLuogo(), ristAggiunto.getLuogo());
        Assert.AreEqual(rist.getIdRistorante(), ristAggiunto.getIdRistorante());
    }

}

```

DEPLOYMENT

ARTEFATTI



DEPLOYMENT TYPE-LEVEL

