Analisi Tecnica

Introduzione generale al progetto

Il progetto nasce con l'obiettivo di gestire in modo ordinato gli elementi tipici di un contesto universitario: studenti, professori e corsi. Oltre alle operazioni base di CRUD (creazione, lettura, aggiornamento e cancellazione), il sistema permette di registrare voti, calcolare medie, assegnare professori ai corsi, tracciare tutte le operazioni eseguite dagli utenti e gestire una coda FIFO per le richieste di iscrizione di nuovi studenti.

In una versione iniziale, tutto veniva gestito in memoria attraverso liste, code e dizionari. Con l'evoluzione dell'applicativo, si è aggiunta una componente di persistenza tramite database SQL Server, in particolare per la configurazione del logger e per il salvataggio dei log.

Il database e la scelta della persistenza

Il database attualmente contiene due tabelle principali:

LoggerConfig – memorizza la configurazione del sistema di logging, cioè se il log è attivo, quali categorie devono essere registrate, e se utilizzare file, database o entrambi.

LogEvents – contiene gli eventi di log generati dall'applicazione.

Gli script SQL utilizzati per creare le tabelle sono i seguenti:

```
CREATE TABLE LoggerConfig (

    Id INT IDENTITY(1,1) PRIMARY KEY,

    Attivo BIT NOT NULL,

    LogStudenti BIT NOT NULL,

    LogProfessori BIT NOT NULL,
```

```
    LogCorsi BIT NOT NULL,
    LogCodeStorico BIT NOT NULL,
    UsaFile BIT NOT NULL,
    UsaDatabase BIT NOT NULL,
    FilePath NVARCHAR(500) NULL
);

CREATE TABLE LogEvents (
    Id INT IDENTITY(1,1) PRIMARY KEY,
    Timestamp DATETIME NOT NULL,
    Categoria NVARCHAR(50) NOT NULL,
    Messaggio NVARCHAR(MAX) NOT NULL
);
```

Il resto (studenti, professori, corsi, voti, code, storico) è ancora gestito in memoria. Questo perché nasceva dal prototipo originale senza database; solo alcune parti sono state migrate verso la persistenza per necessità specifiche, come la configurazione e i log.

La connessione al database e il pattern Singleton

La classe che gestisce la connessione SQL al database segue il **pattern Singleton**, cioè ne esiste una sola istanza per tutta l'applicazione. Questo approccio è utile quando vogliamo:

- evitare di ricreare connessioni inutilmente,
- centralizzare la gestione della stringa di connessione,
- garantire che ogni parte del codice utilizzi lo stesso punto di accesso.

Nel progetto è stata implementata una classe `ConnessioneSingleton` che espone un metodo `CreateConnection()` per ottenere una nuova `SqlConnection` basata sulla stessa connessione configurata.

Il vantaggio è la semplicità: ogni repository può ottenere una connessione con una riga di codice, assicurandosi che la configurazione sia coerente.

Il sistema di logging

Il logging è una parte fondamentale dell'app. È stato progettato per essere:

- configurabile,
- suddiviso in categorie (studenti, professori, corsi, storico),
- salvabile su file, su database o su entrambi.

La configurazione viene caricata dal database tramite `LoggerRepository`, che esegue query SQL e restituisce un oggetto `LoggerConfig`.

La classe responsabile del logging vero e proprio è `LogService`, anch'essa implementata come Singleton per permettere una gestione centralizzata del log.

Quando il controller esegue qualsiasi operazione, chiama:

LogService.Instance.Log("studenti", msg);

Il servizio verifica nella `LoggerConfig` se quella categoria è abilitata e, in base alle impostazioni, salva il messaggio su file, su database, o su entrambi.

Le entità principali

Studente – contiene nome, cognome, matricola, corso di iscrizione e la media calcolata. Implementa `IComparable` per permettere un ordinamento naturale degli studenti in base alla media e, in caso di parità, in base al cognome e al nome.

Professore – contiene i dati personali e la materia insegnata; può essere associato a più corsi.

CorsoDiLaurea – ha un codice, un nome e una lista di professori e materie associate.

LoggerConfig – rappresenta la configurazione del logger caricata dal database.

Logger (evento) – rappresenta un singolo evento di log salvato nel database.

StoricoOperazioni – registra le operazioni dell'utente in una lista interna, permettendo anche di annullare l'ultima operazione.

CodaIscrizioni – implementa una coda FIFO per le richieste di nuovi studenti.

Il pattern MVC nel progetto

Model: include tutte le entità come Studente, Professore, Corso, LoggerConfig, LogEvents, StoricoOperazioni e CodaIscrizioni. Tutta la logica dei dati sta qui.

View: nel progetto attuale è rappresentata dalla console. Tutte le stampe sono effettuate dalla parte esterna del programma.

Controller: `UniversitàController` è il punto di accesso principale. Riceve richieste dall'esterno, effettua operazioni sui repository, registra operazioni nello storico, invoca il logger e restituisce il risultato.

Repository: contengono la logica per gestire i dati. Ne esistono due tipi:

- repository "in memoria" (per studenti, professori, corsi),

- repository "persistenti" come il `LoggerRepository`.

Il controller e il flusso delle operazioni

Il `UniversitàController` combina insieme repository, storico e logging. Ogni operazione segue infatti una logica abbastanza costante:

1. chiama il repository per eseguire l'azione richiesta;
2. registra l'operazione nello storico;
3. invia un messaggio al sistema di logging.

Un esempio è l'aggiunta di uno studente:

```
bool ok = repo.AggiungiStudente(nome, cognome, matricola, codiceCorso);

string msg = $"Aggiunto studente {nome} {cognome}";

if (ok)

    storico.Registra(msg);

LogService.Instance.Log("studenti", msg);
```

Obiettivi principali del sistema

Gli obiettivi richiesti dal progetto sono stati tutti raggiunti:

- gestione CRUD completa per studenti, corsi e professori;
- registrazione voti e calcolo delle medie;
- tracciamento delle operazioni utente tramite storico;
- logging configurabile con salvataggio su file, database o entrambi;
- gestione delle richieste tramite coda FIFO.
- Utilizzo del pattern MVC
- Utilizzo del pattern Singleton