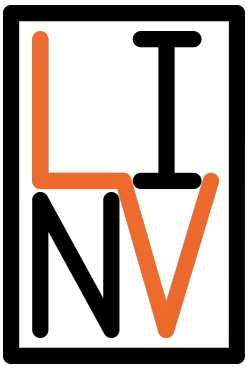

LINV Team

Specifica Tecnica

Progetto di ingegneria del software
A.A 2022/2023

Informazioni	
Versione	1.0
Uso	Esterno
Data	18/06/2023
Destinatari	LINV Team
	Socomec
	Tullio Vardanega
	Riccardo Cardin
Responsabile	Matteo Cusin
Amministratore	Riccardo Rossi
Verificatori	Alessandro Baldissera
	Alberto Casado Moreno
	Matteo Cusin
	Riccardo Rossi
Redattori	Alessandro Baldissera
	Mauro Carnuccio
	Alberto Casado Moreno
	Matteo Cusin
	Nicola Ravagnan
	Riccardo Rossi
	Alessandro Santin

Indice

Elenco delle figure

Elenco delle tabelle

Registro delle modifiche

Ver.	Data	Autore	Ruolo	Verificatore	Descrizione
1.0	18/06/2023	Matteo Cusin	Responsabile		Approvazione documento
0.15	13/06/2023	Alessandro Santin	Progettista	Matteo Cusin	Aggiunti design pattern utilizzati
0.14	10/06/2023	Alberto Casado Moreno	Progettista	Nicola Ravagnan	Aggiunta la struttura dell'architettura
0.13	03/06/2023	Riccardo Rossi	Progettista	Matteo Cusin	Modificata architettura Front-end SmartLogStatistics
0.12	03/06/2023	Matteo Cusin	Progettista	Alberto Casado Moreno	Aggiunte query database SmartLogStatistics
0.11	18/05/2023	Mauro Carnuccio, Alberto Casado Moreno, Nicola Ravagnan	Progettista	Alessandro Baldissera	Aggiunta architettura Back-end SmartLogStatistics
0.10	13/05/2023	Riccardo Rossi	Progettista	Matteo Cusin	Aggiunta architettura Front-end SmartLogStatistics
0.9	07/05/2023	Riccardo Rossi	Progettista	Alessandro Baldissera	Modificata architettura Front-end SmartLogViewer
0.8	28/04/2023	Matteo Cusin	Progettista	Nicola Ravagnan	Aggiunto middleware per ricerca sequenze SmartLogViewer
0.7	20/04/2023	Matteo Cusin	Progettista	Alberto Casado Moreno	Aggiunto middle-ware SmartLogViewer
0.6	16/04/2023	Riccardo Rossi	Progettista	Alessandro Baldissera	Aggiunta architettura Front-end SmartLogViewer

0.5	14/04/2023	Mauro Carnuccio, Alberto Casado Moreno, Nicola Ravagnan	Progettista	Riccardo Rossi	Aggiunta API SmartLogStatistics
0.4	05/04/2023	Mauro Carnuccio, Riccardo Rossi	Progettista	Matteo Cusin	Aggiunta API SmartLogViewer
0.3	04/04/2023	Mauro Carnuccio, Matteo Cusin	Progettista	Nicola Ravagnan	Definizione struttura documento, aggiunta sezione di libreria core
0.2	30/03/2023	Mauro Carnuccio	Progettista	Alessandro Baldissera	Aggiunta sezione di database
0.1	20/03/2023	Alessandro Baldissera	Progettista	Riccardo Rossi	Definizione struttura documento

1. Introduzione

1.1 Scopo del documento

Il documento ha lo scopo di definire la struttura architetturale e il design di dettaglio delle applicazioni *SmartLogViewer* e *SmartLogStatistics* descrivendo: le tecnologie utilizzate, le API e l'architettura dei due applicativi.

1.2 Glossario

Questo documento, come tutti gli altri stilati durante la realizzazione del progetto, è corredato da un ***Glossario*** che si può trovare allegato alla documentazione, nel quale si definiscono tutti i termini specifici al progetto o di significato ambiguo. Quando un termine è definito nel ***Glossario*** si trova una G a pedice del termine stesso.

1.3 Riferimenti

1.3.1 Riferimenti normativi

- ***Way of Working***;
- Regolamento del progetto didattico: https://www.math.unipd.it/~tullio/IS-1/2022/Dispense/PD02.pdf.

1.3.2 Riferimenti informativi

- Capitolato C5: https://www.math.unipd.it/~tullio/IS-1/2022/Progetto/C5.pdf;
- ***Analisi dei Requisiti***;
- ***Verbali interni***;
- ***Verbali esterni***;
- Progettazione, le dipendenze fra le componenti: https://www.math.unipd.it/ rcardin/swea/2023/Object-Oriented%20Progamming%20Principles%20Revised.pdf;
- Progettazione e programmazione, diagrammi delle classi (UML): https://www.math.unipd.it/ rcardin/swea/2023/Diagrammi%20delle%20Classi.pdf;

- Progettazione, i pattern architetturali: https://www.math.unipd.it/ rcardin/swea/2022/Software%20Architecture%20Patterns.pdf;
- Progettazione, il pattern Dependency Injection: https://www.math.unipd.it/ rcardin/swea/2022/Design%20Pattern%20Architetturali%20-%20Dependency%20Injection.pdf;
- Progettazione, il pattern Model-View-Controller e derivati: https://www.math.unipd.it/~rcardin/sweb/2022/L02.pdf;
- Progettazione, i pattern creazionali (GoF): https://www.math.unipd.it/ rcardin/swea/2022/Design%20Pattern%20Creazionali.pdf;
- Progettazione software: https://www.math.unipd.it/~tullio/IS-1/2022/Dispense/T07.pdf;
- Progettazione, i pattern strutturali (GoF): https://www.math.unipd.it/ rcardin/swea/2022/Design%20Pattern%20Strutturali.pdf;
- Progettazione, i pattern di comportamento (GoF): https://www.math.unipd.it/ rcardin/swea/2021/Design%20Pattern%20Comportamentali_4x4.pdf;
- Programmazione, SOLID programming: https://www.math.unipd.it/ rcardin/swea/2021/SOLID%20Principles%20of%20Object-Oriented%20Design_4x4.pdf;.

2. Tecnologie

2.1 Introduzione

La seguente sezione elenca in modo esaustivo le tecnologie scelte per l'implementazione di entrambe le applicazioni richieste dal capitolato.

2.2 Elenco delle tecnologie

Nome	Versione	Descrizione
Linguaggi		
C#	11	È un linguaggio di programmazione orientato agli oggetti, creato e mantenuto da *Microsoft*
CSS	3	È il linguaggio di definizione di stili standard del web mantenuto dal consorzio *W3C*
HTML	5	È il linguaggio di markup standard per la creazione di pagine web mantenuto dal consorzio *W3C*
Typescript	8.19.3	È il linguaggio di programmazione orientato agli oggetti e fortemente tipizzato per agevolare la scrittura di codice Javascript. Compila nello standard ECMAScript 6, producendo codice adatto all'uso su pagine web
Framework		
.NET	6.0 LTS	Framework per la creazione di applicazioni desktop di *Microsoft*
Angular	14.0	Framework per la creazione di single-page application per browser
Bootstrap	5.2.3	Framework per componenti grafiche di pagine web
Entity Framework	EF Core 7.0	Framework open-source object-relational mapping(ORM)
Jasmine	4.5.0	Framework open-source per il testing di codice Javascript
Karma	6.4.0	Strumento che genera un server web che esegue il codice di test javascript per ogni browser connesso
Librerie		
CsvHelper	30.0.0	Libreria per la lettura e la scrittura di un file CSV
d3.js	7.8.4	Libreria per la creazione di grafici svg
Npgsql	7.0.1	Libreria per l'accesso a PostgresSQL
Strumenti		
Compodoc	1.1.19 - 1.1.21	Strumento open-source per la generazione di documentazione per Angular

Jetbrains Rider	2022	IDE non ufficiale per progetti C# e .NET distribuito da *Jetbrains* e interoperabile con Visual Studio Express e MSBuild; compatibile con sistemi Windows, Mac e Linux
MSBuild	17.5	Build tool per la compilazione dei sorgenti gestito in autonomia dall'IDE
PostgresSQL	15	Database relazionale open source basato su SQL
Visual Studio	Express 2022	IDE ufficiale per progetti C# e .NET distribuito da *Microsoft*

Tabella 2.1: Elenco esaustivo delle tecnologie adottate

3. API

3.1 SmartLogViewer

3.1.1 Parse

Descrizione: Questo endpoint REST permette di inviare un file CSV e avere come risposta, in caso di esito positivo, un file JSON con i dati convertiti.

- **endpoint**: `/api/parse`;
- **Metodo HTTP**: `POST`;
- **body**: formato `multipart/form-data`;

Esito	Codice HTTP	Body	Descrizione
Positivo	**201**	JSON(come da descrizione)	Viene caricato un file ed elaborato con successo
Negativo	**400**	`{ "code": 1, "message": "..." }`	Viene caricato un file con almeno una linea non strutturata correttamente, il messaggio conterrà la specifica dell'errore
Negativo	**400**	`{ "code": 2, "message": "..." }`	Viene caricato un file che ha almeno un dato dal formato non corretto, il messaggio conterrà la specifica dell'errore

Tabella 3.1: Esiti HTTP della chiamata `POST /api/parse`.

3.1.2 Sequences

3.1.2.1 GET nomi delle sequenze

Descrizione: Questo endpoint REST permette di prelevare i nomi delle sequenze in formato JSON.

- **endpoint**: `/api/sequences`;
- **Metodo HTTP**: `GET`;

Esito	Codice HTTP	Body	Descrizione
Positivo	**200**	JSON(come da descrizione)	Viene ritornato l'insieme dei nomi delle sequenze di eventi note disponibili
Negativo	**500**	-	La lettura del file non è avvenuta correttamente

Tabella 3.2: Esiti HTTP della chiamata GET `/api/sequences`.

3.1.2.2 GET informazioni di una sequenza

Descrizione: Questo endpoint REST permette di prelevare le informazioni di una determinata sequenza di eventi in formato JSON.

- **endpoint**: `/api/sequences/{sequenceName}`;
- **Metodo HTTP**: `GET`;
- **Parametri**:
  - `sequenceName`: nome della sequenza di cui prelevare i dati.

Esito	Codice HTTP	Body	Descrizione
Positivo	**200**	JSON(come da descrizione)	Vengono prelevate le informazioni con successo
Negativo	**404**	{ "sequenceName" }	Non è stata trovata la sequenza richiesta
Negativo	**500**	-	Il server ha rilevato una condizione imprevista per la quale non è riuscito a soddisfare la richiesta

Tabella 3.3: Esiti HTTP della chiamata `GET /api/sequences/{sequenceName}`.

3.2 SmartLogStatistics

3.2.1 Upload

3.2.1.1 POST caricamento file

Descrizione: Questo endpoint REST permette di caricare file di log nel database.

- **endpoint**: `/api/upload`;
- **Metodo HTTP**: `POST`;

- **Content-type**: `multipart/form-data`;

Esito	Codice HTTP	Body	Descrizione
Positivo	**201**	-	Vengono caricati i dati nel database con successo
Negativo	**400**	`{ "code": 1, "message": "..." }`	Si è cercato di caricare uno o più file che hanno almeno una linea non strutturata correttamente, il messaggio conterrà la specifica dell'errore
Negativo	**400**	`{ "code": 2, "message": "..." }`	Si è cercato di caricare uno o più file che hanno almeno un dato dal formato non corretto, il messaggio conterrà la specifica dell'errore
Negativo	**409**	`{ "code": 6, "message": "..." }`	Si è cercato di caricare uno o più file che sono già presenti all'interno del database, il messaggio conterrà la specifica dell'errore
Negativo	**500**	`{ "code": 4, "message": "..." }`	Il server ha rilevato una condizione imprevista per la quale non è riuscito a soddisfare la richiesta

Tabella 3.4: Esiti HTTP della chiamata `POST /api/upload`.

3.2.2 Statistics

3.2.2.1 GET statistiche con intervallo temporale

Descrizione: Questo endpoint REST permette di prelevare le statistiche di un file di log, dal database, comprese nell'intervallo temporale dato. Le statistiche ritornate sono in formato JSON.

- Ritorno:
  - Numero di storici analizzati;
  - Media di eventi per file di log;
  - Massimo numero di eventi per file di log;
  - Deviazione standard sul numero di eventi per file di log.
- **endpoint**: `/api/statistics/{startDateTime}/{endDateTime}`;
- **Metodo HTTP**: `GET`;
- **Parametri**:
  - `startDateTime`: data dell'UPS del primo file di log da prelevare;

– `endDateTime`: data dell'UPS del ultimo file di log da prelevare.

Esito	Codice HTTP	Body	Descrizione
Positivo	**200**	JSON(come da descrizione)	Vengono prelevate le statistiche dal database con successo
Negativo	**400**	`{ "code": 3, "message": "..." }`	Vengono inseriti come parametri delle date non compatibili fra loro
Negativo	**404**	`{ "code": 5, "message": "..." }`	Il risultato della richiesta è vuoto
Negativo	**500**	`{ "code": 4, "message": "..." }`	Il server ha rilevato una condizione imprevista per la quale non è riuscito a soddisfare la richiesta

Tabella 3.5: Esiti HTTP della chiamata `GET /api/statistics/{startDateTime}/{endDateTime}`.

3.2.3 Info

3.2.3.1 GET lista di codici e descrizioni

Descrizione: Questo endpoint REST permette di prelevare la lista dei codici e delle loro descrizioni dal database. Le statistiche ritornate sono in formato JSON.

- Ritorno per ogni evento:
  - Codice;
  - Descrizione;
- **endpoint**: `/api/info/code-description`;
- **Metodo HTTP**: `GET`;

Esito	Codice HTTP	Body	Descrizione
Positivo	**200**	JSON(come da descrizione)	Vengono prelevate le informazioni dal database con successo
Negativo	**404**	`{ "code": 5, "message": "..." }`	Il risultato della richiesta è vuoto
Negativo	**500**	`{ "code": 4, "message": "..." }`	Il server ha rilevato una condizione imprevista per la quale non è riuscito a soddisfare la richiesta

Tabella 3.6: Esiti HTTP della chiamata `GET /api/info/code-description`.

3.2.3.2 GET coppia di timestamp del primo e dell'ultimo log cronologicamente

Descrizione: Questo endpoint REST permette di prelevare l'intervallo temporale dei log dall'intero database. Le statistiche ritornate sono in formato JSON.

- Ritorno:
  - Il primo momento in cui si è verificato un evento;
  - L'ultimo momento in cui si è verificato un evento;
- **endpoint**: `/api/info/timeinterval`;
- **Metodo HTTP**: `GET`;

Esito	Codice HTTP	Body	Descrizione
Positivo	**200**	JSON(come da descrizione)	Vengono prelevate le informazioni dal database con successo
Negativo	**404**	`{ "code": 5, "message": "..." }`	Il risultato della richiesta è vuoto
Negativo	**500**	`{ "code": 4, "message": "..." }`	Il server ha rilevato una condizione imprevista per la quale non è riuscito a soddisfare la richiesta

Tabella 3.7: Esiti HTTP della chiamata `GET /api/info/timeinterval`.

3.2.3.3 GET lista dei firmware

Descrizione: Questo endpoint REST permette di prelevare la lista di firmware dall'intero database. Le statistiche ritornate sono in formato JSON.

- Ritorno:
  - Lista di firmware.
- **endpoint**: `/api/info/firmwarelist`;
- **Metodo HTTP**: `GET`;

Esito	Codice HTTP	Body	Descrizione
Positivo	**200**	JSON(come da descrizione)	Vengono prelevate le informazioni dal database con successo
Negativo	**404**	`{ "code": 5, "message": "..." }`	Il risultato della richiesta è vuoto
Negativo	**500**	`{ "code": 4, "message": "..." }`	Il server ha rilevato una condizione imprevista per la quale non è riuscito a soddisfare la richiesta

Tabella 3.8: Esiti HTTP della chiamata `GET /api/info/firmwarelist`.

3.2.4 Data

3.2.4.1 GET raggruppamento per campi specifici con intervallo temporale

Descrizione: Questo endpoint REST permette di prelevare gli eventi nel database, compresi nell'intervallo temporale dato, raggruppati per i campi specificati, con al relativa frequenza di occorrenza. Restituisce le relative statistiche in formato JSON.

- **endpoint**:

  `/api/data/frequency/{start-DateTime}/{end-DateTime}?d={dataBool}&f={firmwareBool}&u={unitBool}&s={subunitBool}`;

- **Parametri:**

  - `start-DateTime`: data dell'UPS del primo file di log da prelevare;
  - `end-DateTime`: data dell'UPS del ultimo file di log da prelevare.

Nome	Tipo	Descrizione	Valore di default
`d`	`boolean`	Se `true` viene applicato un raggruppamento per il campo `Data`	false
`f`	`boolean`	Se `true` viene applicato un raggruppamento per il campo `Firmware`	false
`u`	`boolean`	Se `true` viene applicato un raggruppamento per il campo `Unit`	false
`s`	`boolean`	Se `true` viene applicato un raggruppamento per il campo `SubUnit`	false

Tabella 3.9: Parametri della chiamata `GET /api/data/frequency/{start-DateTime}/{end-DateTime}`

- **Metodo HTTP**: `GET`;

Esito	Codice HTTP	Body	Descrizione
Positivo	**200**	JSON(come da descrizione)	Vengono restituite le statistiche con successo
Negativo	**400**	`{ "code": 3, "message": "..." }`	Vengono inseriti come parametri delle date non compatibili fra loro
Negativo	**404**	`{ "code": 5, "message": "..." }`	Il risultato della richiesta è vuoto
Negativo	**500**	`{ "code": 4, "message":"..." }`	Il server ha rilevato una condizione imprevista per la quale non è riuscito a soddisfare la richiesta

Tabella 3.10: Esiti HTTP della chiamata `GET /api/data/frequency/{start-DateTime}/{end-DateTime}?d={dataBool}&f={firmwareBool}&u={unitBool}&s={subunitBool}`.

3.2.4.2 GET dati per grafico cumulativo

Descrizione: Questo endpoint REST permette di prelevare i `DateTime` compresi nell'intervallo temporale dato in cui si è verificato l'evento specificato tramite il code. Il file ritornato è in formato JSON.

- **endpoint**:
  `/api/data/cumulative/{start-DateTime}/{end-DateTime}/{code}`;
- **Parametri:**
  - `code`: Code dell'evento di cui si vogliono avere le occorenze;
  - `start-DateTime`: data dell'UPS del primo file di log da prelevare;
  - `end-DateTime`: data dell'UPS del ultimo file di log da prelevare.
- **Metodo HTTP**: `GET`;

Esito	Codice HTTP	Body	Descrizione
Positivo	**200**	JSON(come da descrizione)	Vengono restituite le statistiche con successo
Negativo	**400**	`{ "code": 3, "message": "..." }`	Vengono inseriti come parametri delle date non compatibili fra loro
Negativo	**404**	`{ "code": 5, "message": "..." }`	Il risultato della richiesta è vuoto
Negativo	**500**	`{ "code": 4, "message":"..." }`	Il server ha rilevato una condizione imprevista per la quale non è riuscito a soddisfare la richiesta

Tabella 3.11: Esiti HTTP della chiamata `GET /api/data/cumulative/{start-DateTime}/{end-DateTime}/{code}`.

3.2.4.3 Get dati per istogramma

Descrizione: Questo endpoint REST permette di prelevare gli eventi e il numero di occorrenze, compresi nell'intervallo temporale dato. Il file ritornato è in formato JSON.

- **endpoint**:
  `/api/data/totalbycode/{start-DateTime}/{end-DateTime}`;
- **Parametri:**
  - `start-DateTime`: data dell'UPS del primo file di log da prelevare;
  - `end-DateTime`: data dell'UPS del ultimo file di log da prelevare.
- **Metodo HTTP**: `GET`;

Esito	Codice HTTP	Body	Descrizione
Positivo	**200**	JSON(come da descrizione)	Vengono restituite le statistiche con successo
Negativo	**400**	`{ "code": 3, "message": "..." }`	Vengono inseriti come parametri delle date non compatibili fra loro
Negativo	**404**	`{ "code": 5, "message": "..." }`	Il risultato della richiesta è vuoto
Negativo	**500**	`{ "code": 4, "message":"..." }`	Il server ha rilevato una condizione imprevista per la quale non è riuscito a soddisfare la richiesta

Tabella 3.12: Esiti HTTP della chiamata `GET /api/data/totalbycode/{start-DateTime}/{end-DateTime}`.

3.2.4.4 GET dati grafico a torta

Descrizione: Questo endpoint REST permette di prelevare il numero di occorrenze, raggruppati per versione firmware, compresi nell'intervallo temporale dato in cui si è verificato l'evento specificato tramite il code. Il file ritornato è in formato JSON.

- **endpoint**:
  `/api/data/totalbyfirmware/{start-DateTime}/{end-DateTime}/{code}`;
- **Parametri:**
  - `code`: code dell'evento di cui si vogliono avere le occorenze;
  - `start-DateTime`: data dell'UPS del primo file di log da prelevare;
  - `end-DateTime`: data dell'UPS del ultimo file di log da prelevare.
- **Metodo HTTP**: `GET`;

Esito	Codice HTTP	Body	Descrizione
Positivo	**200**	JSON(come da descrizione)	Vengono restituite le statistiche con successo
Negativo	**400**	`{ "code": 3, "message": "..." }`	Vengono inseriti come parametri delle date non compatibili fra loro
Negativo	**404**	`{ "code": 5, "message": "..." }`	Il risultato della richiesta è vuoto
Negativo	**500**	`{ "code": 4, "message":"..." }`	Il server ha rilevato una condizione imprevista per la quale non è riuscito a soddisfare la richiesta

Tabella 3.13: Esiti HTTP della chiamata `GET /api/data/totalbyfirmware/{start-DateTime}/{end-DateTime}/{code}`.

4. Architettura

4.1 Diagrammi delle classi

4.1.1 Libreria Core

Le dipendenze della libreria Core sono:

- `CSVHelper`

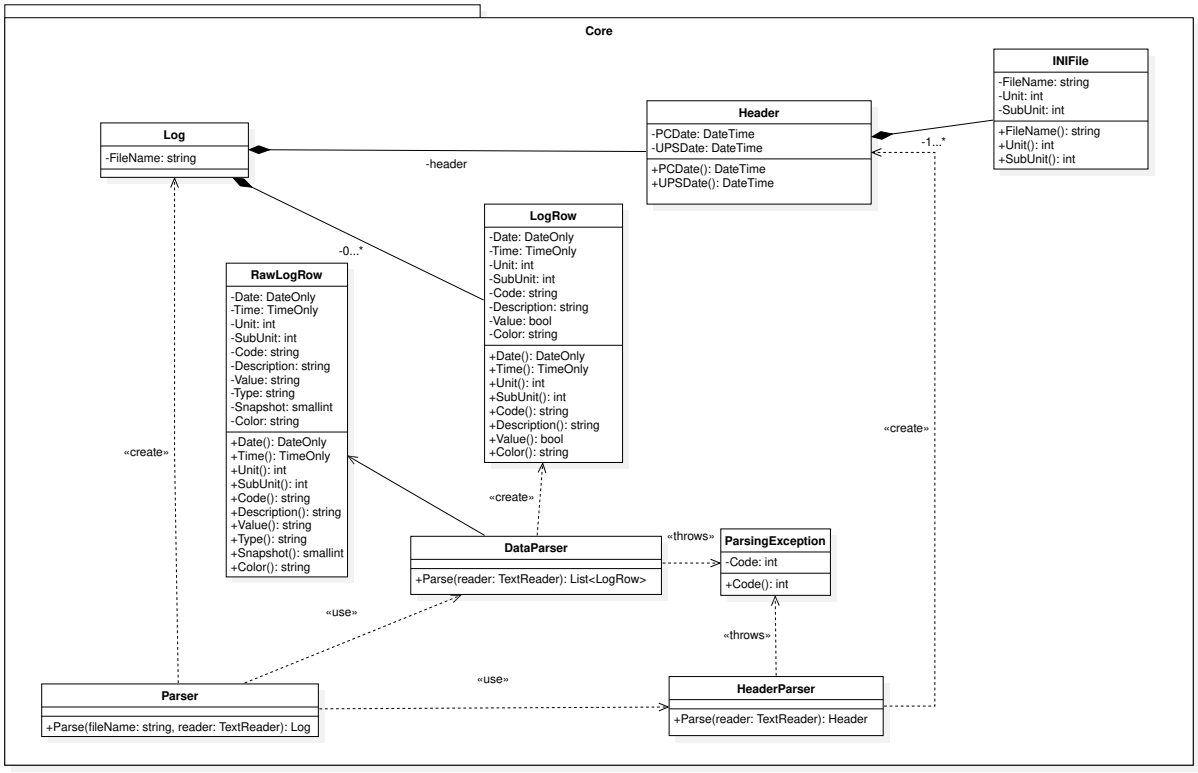

Figura 4.1: Diagramma delle classi del parser.

La libreria Core soddisfa i seguenti requisiti:

- **RFV-1.1**: l'utente vuole caricare un file di log;
- **RFV-1.1.1**: l'utente visualizza un messaggio positivo in caso di successo nel caricamento di un file di log;
- **RFV-1.2**: l'utente ha provato a caricare un file impossibile da leggere correttamente e quindi visualizza un messaggio di errore;

- **RFV-1.2.1**: l'utente visualizza un errore dato dal formato non corretto di almeno una linea del file;
- **RFV-1.2.2**: l'utente visualizza un errore generato dai dati del file che non si presentano nel formato corretto;
- **RFS-1.2**: l'utente ha selezionato uno o più file impossibili da leggere correttamente e quindi visualizza un messaggio di errore;
- **RFS-1.2.1**: l'utente ha caricato uno o più file aventi un formato non corretto e quindi visualizza un messaggio di errore;
- **RFS-1.2.2**: l'utente ha caricato uno o più file aventi alcuni dati che presentano un formato non corretto e quindi visualizza un messaggio di errore;
- **RV-1.1**: il prodotto deve essere in grado di analizzare file CSV.

I componenti della libreria Core sono:

4.1.1.1 Parser

- **Metodi**
  - `Parse(fileName: String, reader: TextReader): Log`

    Descrizione:
    * Esegue la conversione di un file di log in formato CSV in un oggetto di tipo `Log`.

    Input:
    * `fileName`: stringa contenente il nome del file da convertire;
    * `reader`: oggetto di tipo `TextReader` che rappresenta il contenuto del file di log come un flusso di testo.

    Output:
    * Oggetto di tipo `Log` che rappresenta l'intero file di log convertito.

4.1.1.2 HeaderParser

- **Metodi**
  - `Parse(reader: TextReader): Header`

    Descrizione:
    * Esegue la conversione dell'intestazione del file di log e lo converte in un oggetto di tipo `Header`.

    Input:
    * `reader`: oggetto di tipo `TextReader` contenente l'intestazione del file di log.

Output:

* Oggetto di tipo `Header` che rappresenta l'intestazione del file di log.

Eccezioni:

* `ParsingException`.

4.1.1.3 DataParser

- **Metodi**
  - `Parse(reader: TextReader): List<LogRow>`

    Descrizione:

    * Esegue la conversione del corpo del file di log in una lista di oggetti di tipo `LogRow`.

    Input:

    * `reader`: oggetto di tipo `TextReader` dell'intestazione del file di log.

    Output:

    * Oggetto di tipo `Header` che rappresenta l'intestazione del file di log.

    Eccezioni:

    * `ParsingException`.

4.1.1.4 ParsingException

- **Attributi**
  - `Code`: intero che indica il codice dell'errore che si è verificato (il codice **1** è associato all'errore **invalidFormat**, il codice **2** è associato all'errore **invalidData** che rappresentano rispettivamente una situazione in cui vi è un errore sul formato dei dati letti ed una situazione in cui vi è un errore sul dominio dei dati letti).

4.1.1.5 Log

- **Attributi**
  - `FileName`: stringa che indica il nome del file dal quale è creato l'oggetto `Log`;
  - `Header`: oggetto di tipo `Header` che rappresenta l'intestazione del file di log;
  - `Data`: oggetto di tipo `List<LogRow>` che rappresenta gli eventi contenuti nel file di log.

4.1.1.6 Header

- **Attributi**
  - `PCDate`: oggetto di tipo `DateTime` che rappresenta il campo *PCDate* del file di log;
  - `UPSDate`: oggetto di tipo `DateTime` che rappresenta il campo *UPSDate* del file di log;
  - `INIFile`: oggetto di tipo `List<INIFile>` che rappresenta l'insieme dei file con estensione *.ini* associati al file di log.

4.1.1.7 INIFile

- **Attributi**
  - `FileName`: stringa che indica il nome del file *.ini*;
  - `Unit`: variabile intera che rappresenta l'unità a cui è associato il file;
  - `SubUnit`: variabile intera che rappresenta la sotto-unità a cui è associato il file.

4.1.1.8 LogRow

- **Attributi**
  - `Date`: oggetto di tipo `DateOnly` che rappresenta la data in cui si è registrato l'evento;
  - `Time`: oggetto di tipo `TimeOnly` che rappresenta l'orario in cui si è registrato l'evento;
  - `Unit`: variabile intera che rappresenta l'unità in cui si è registrato l'evento;
  - `SubUnit`: variabile intera che rappresenta la sotto-unità in cui si è registrato l'evento;
  - `Code`: stringa che indica il codice identificativo dell'evento registrato;
  - `Description`: stringa che indica la descrizione dell'evento registrato;
  - `Value`: variabile booleana che indica lo stato dell'evento registrato;
  - `Color`: stringa che indica un colore associato all'evento registrato.

4.1.1.9 RawLogRow

- **Attributi**
  - `Date`: oggetto di tipo `DateOnly` che rappresenta la data in cui si è registrato l'evento;
  - `Time`: oggetto di tipo `TimeOnly` che rappresenta l'orario in cui si è registrato l'evento;
  - `Unit`: variabile intera che rappresenta l'unità in cui si è registrato l'evento;
  - `SubUnit`: variabile intera che rappresenta la sotto-unità in cui si è registrato l'evento;

- `Code`: stringa che indica il codice identificativo dell'evento registrato;
- `Description`: stringa che indica la descrizione dell'evento registrato;
- `Value`: stringa che indica lo stato dell'evento registrato;
- `Type`: stringa che indica il tipo di evento registrato;
- `SnapShot`: variabile di tipo `smallint` che si mappa sul campo *Snapshot* del file di log (uso interno dell'azienda proponente);
- `Color`: stringa che indica un colore associato all'evento registrato.

4.1.2 Backend SmartLogViewer

Le dipendenze del progetto di back-end e middleware `SmartLogViewer` sono:

- Libreria `Core`

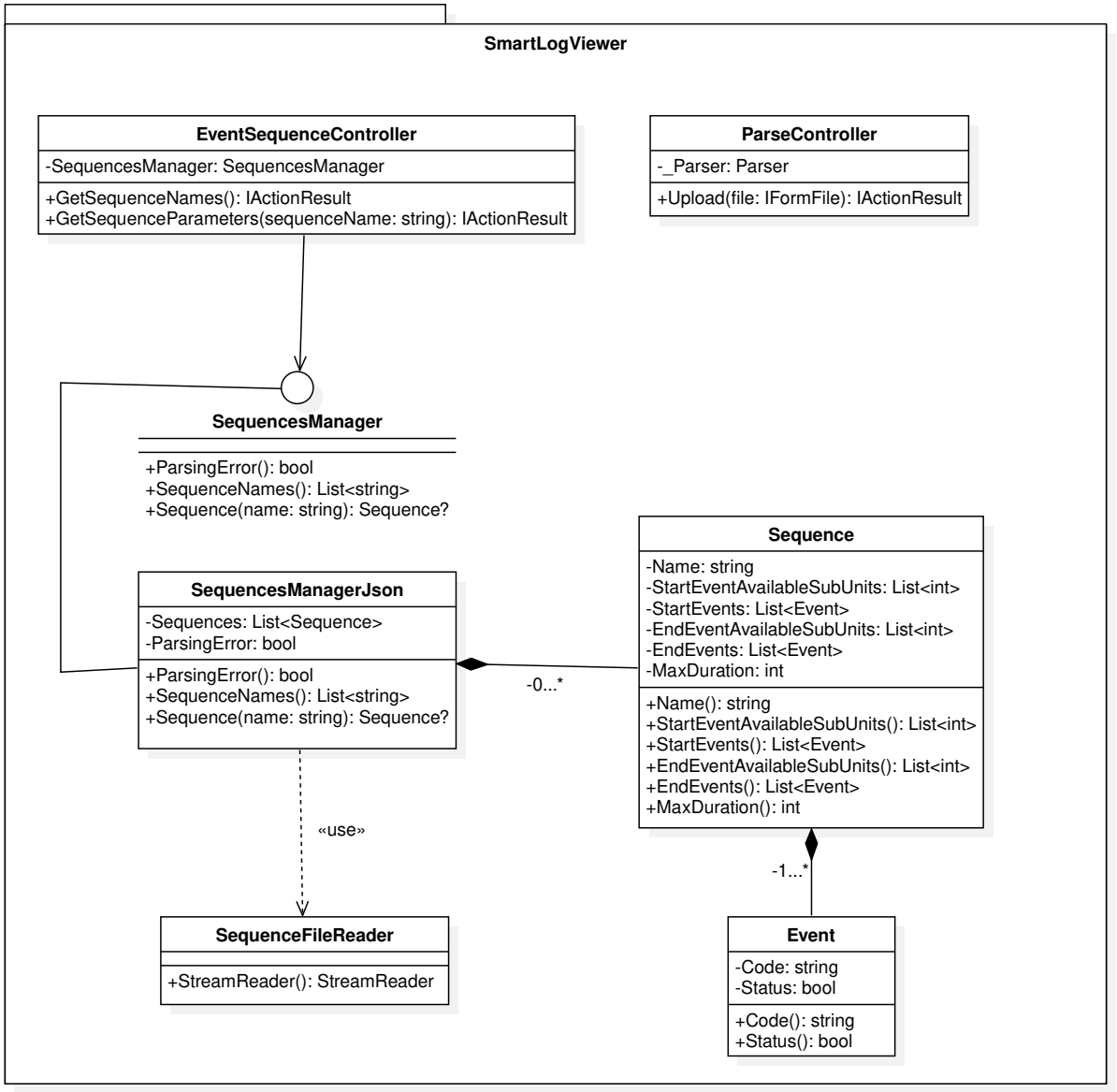

Figura 4.2: Diagramma delle classi del progetto SmartLogViewer.

I componenti del progetto SmartLogViewer sono:

4.1.2.1 ParseController

- **Metodi**
  - `Upload(IFormFile file): IActionResult`

    Descrizione:
    * Gestisce una chiamata *HTTP POST* sull'endpoint `api/parse` e ritorna un oggetto di tipo `IActionResult`.

    Input:
    * `file`: file ottenuto da una richiesta avente *Content-Type* pari a `multipart/form-data`.

    Output:
    * Oggetto di tipo `IActionResult` che rappresenta la risposta *HTTP*, la quale può avere esito positivo o negativo (vedi sezione API SmartLogViewer).

- **Attributi**
  - `_Parser`: oggetto di tipo `Parser` dedicato al parsing dei file in ingresso.

4.1.2.2 EventSequenceController

- **Metodi**
  - `GetSequenceNames(): IActionResult`

    Descrizione:
    * Gestisce una chiamata *HTTP GET* sull'endpoint `api/sequences` e ritorna un oggetto di tipo `IActionResult`.

    Output:
    * Oggetto di tipo `IActionResult` che rappresenta la risposta *HTTP*, la quale può avere esito positivo o negativo (vedi sezione API SmartLogViewer).

  - `GetSequenceParameters(string sequenceName): IActionResult`

    Descrizione:
    * Gestisce una chiamata *HTTP GET* sull'endpoint `api/sequences/{sequenceName}` e ritorna un oggetto di tipo `IActionResult`.

    Input:
    * `sequenceName`: stringa che rappresenta il nome della sequenza di cui ritornare le caratteristiche.

    Output:

  - Oggetto di tipo `IActionResult` che rappresenta la risposta *HTTP*, la quale può avere esito positivo o negativo (vedi sezione API SmartLogViewer).

- **Attributi**
  - `SequencesManager`: oggetto di tipo `SequencesManager` atto a contenere le informazioni delle sequenze note.

4.1.2.3 SequencesManager

- **Metodi**
  - `SequencesNames(): List<string>`

    Descrizione:
    - Ritorna la lista di nomi delle possibili sequenze di eventi note.

    Output:
    - Oggetto di tipo `List<string>` che rappresenta la lista di nomi delle possibili sequenze di eventi note.

  - `Sequence(string name): Sequence?`

    Descrizione:
    - Ritorna le caratteristiche della sequenza di eventi richiesta.

    Input:
    - `name`: stringa che rappresenta il nome della sequenza di cui ritornare le caratteristiche.

    Output:
    - Oggetto di tipo `Sequence` che rappresenta le caratteristiche della sequenza di eventi richiesta se viene trovata la sequenza, altrimenti `null`.

4.1.2.4 SequencesManagerJson

- **Metodi**
  - `SequencesNames(): List<string>`

    Descrizione:
    - Ritorna la lista di nomi delle possibili sequenze di eventi note.

    Output:
    - Oggetto di tipo `List<string>` che rappresenta la lista di nomi delle possibili sequenze di eventi note.

  - `Sequence(string name): Sequence?`

    Descrizione:

∗ Ritorna le caratteristiche della sequenza di eventi richiesta.

Input:

∗ `name`: stringa che rappresenta il nome della sequenza di cui ritornare le caratteristiche.

Output:

∗ Oggetto di tipo `Sequence` che rappresenta le caratteristiche della sequenza di eventi richiesta se viene trovata la sequenza (se il file in cui sono contenute le sequenze è in formato JSON), altrimenti `null`.

- **Attributi**
  - `Sequences`: oggetto di tipo `List<Sequence>` atto a contenere le informazioni delle sequenze note;
  - `ParsingError`: variabile booleana che indica se è avvenuto un errore in fase di lettura del file in cui sono salvate le sequenze di eventi note.

4.1.2.5 Sequence

- **Attributi**
  - `Name`: stringa che rappresenta il nome della sequenza di eventi;
  - `StartEventAvailableSubUnits`: oggetto di tipo `List<int>` che rappresenta l'insieme delle subUnit che possono essere presenti nell'evento di partenza della sequenza;
  - `StartEvents`: oggetto di tipo `List<Event>` che rappresenta gli eventi iniziali della sequenza;
  - `EndEventAvailableSubUnits`: oggetto di tipo `List<int>` che rappresenta l'insieme delle subUnit che possono essere presenti nell'evento finale della sequenza;
  - `EndEvents`: oggetto di tipo `List<Event>` che rappresenta gli eventi finali della sequenza;
  - `MaxDuration`: variabile intera che indica la durata temporale massima (in millisecondi) entro cui devono avvenire l'evento iniziale e finale della sequenza di eventi.

4.1.2.6 Event

- **Attributi**
  - `Code`: stringa che rappresenta il codice dell'evento;
  - `Status`: variabile booleana che rappresenta lo stato dell'evento.

4.1.2.7 SequenceFileReader

- **Metodi**

– `StreamReader(): StreamReader`

Descrizione:

* Ritorna uno stream contenente le configurazioni di tutte le sequenze di eventi note.

Output:

* Oggetto di tipo `StreamReader` che rappresenta le configurazioni di tutte le sequenze di eventi note.

4.1.3 Frontend SmartLogViewer

4.1.3.1 LogService

Questo service rende disponibile il log con tutte le informazioni alle altre componenti. La classe log contiene tutte le informazioni di un file di log: il nome del file, le righe del file e l'header con i relativi INIFile.

- **Architettura:**

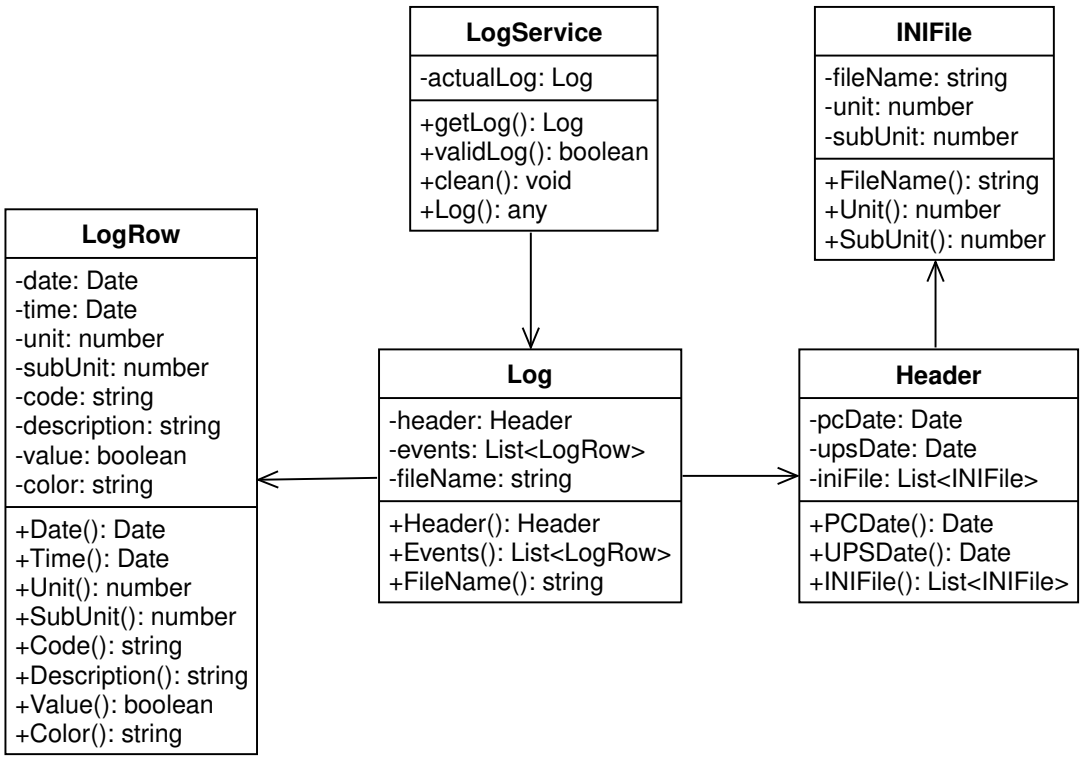

Figura 4.3: Diagramma delle classi del LogService.

– **LogService**
Questo service rende disponibile il log con tutte le informazioni alle altre componenti.

Annotazioni:

* `@Injectable`
  Per poter essere identificato all'interno del sistema di dependency injection di Angular.

Attributi privati:

- `actualLog: Log`
  Il log ricevuto dal backend.

Metodi pubblici:

+ `getLog: Log`
  Ottiene il log salvato nel service;
+ `validLog: boolean`
  Verifica se è presente un log nel service;
+ `clean: void`
  Rimuove il log dal service;
+ `Log: any`
  Assegna il log al service.

– **Log**
Classe che contiene tutte le informazioni di un log.
Attributi privati:

- `header: Header`
  Header del log;
- `events: List<LogRow>`
  Eventi del log;
- `fileName: string`
  Nome del file di log.

Metodi pubblici:

+ `Header(): Header`
  Ottiene l'header del log;
+ `Events(): List<LogRow>`
  Ottiene gli eventi del log;
+ `FileName(): string`
  Ottiene il nome del file di log.

– **Header**
Classe che contiene tutte le informazioni dell'header del log.

Attributi privati:

- `pcDate: Date`
  Data del pc di quando è stato scaricato il log dal macchinario;
- `upsDate: Date`
  Data dell'ups di quando è stato scaricato il log dal macchinario;
- `iniFile: List<INIFile>`
  Lista di INI File che descrivono le componenti del macchinario.

Metodi pubblici:

+ `PCDate() :Date`
  Ottiene la data del pc di quando è stato scaricato il log dal macchinario;
+ `UPSDate() :Date`
  Ottiene la data dell'ups di quando è stato scaricato il log dal macchinario;
+ `INIFile(): List<INIFile>`
  Ottiene la lista di INI File che descrivono le componenti del macchinario.

– **LogRow**
Classe che rappresenta un record del log.

Attributi privati:

– `date: Date`
  Data di registrazione del log;
– `time: Date`
  Ora di registrazione del log;
– `unit: number`
  Unit che ha scatenato l'evento;
– `subUnit: number`
  SubUnit che ha scatenato l'evento;
– `code: string`
  Code dell'evento;
– `description: string`
  Descrizione dell'evento scatenato;
– `value: boolean`
  Valore dell'evento;
– `color: string`
  Colore associato all'evento.

Metodi pubblici:

+ `Date(): Date`
  Ottiene la data di registrazione del log;
+ `Time(): Date`
  Ottiene l'ora di registrazione del log;
+ `Unit(): number`
  Ottiene la Unit che ha scatenato l'evento;
+ `SubUnit(): number`
  Ottiene la SubUnit che ha scatenato l'evento;
+ `Code(): string`
  Ottiene il Code dell'evento;
+ `Description(): string`
  Ottiene la descrizione dell'evento scatenato;
+ `Value(): boolean`
  Ottiene il valore dell'evento;
+ `Color(): string`
  Ottiene il colore associato all'evento.

– **INIFile**
Classe che identifica un INIFile contenuto nell'header di un log.
Attributi privati:

– `fileName: string`
Nome del file INI;

– `unit: number`
Unit associata al file INI;

– `subUnit: number`
SubUnit associata al file INI.

Metodi pubblici:

+ `FileName(): string`
Ottiene il nome del file INI;

+ `Unit(): number`
Ottiene la Unit associata al file INI;

+ `SubUnit(): number`
Ottiene la SubUnit associata al file INI.

4.1.3.2 LogManipulator

Questa interfaccia definisce come presentare i dati alla tabella e al grafico. Viene utilizzato il *command pattern*.

- **Architettura:**

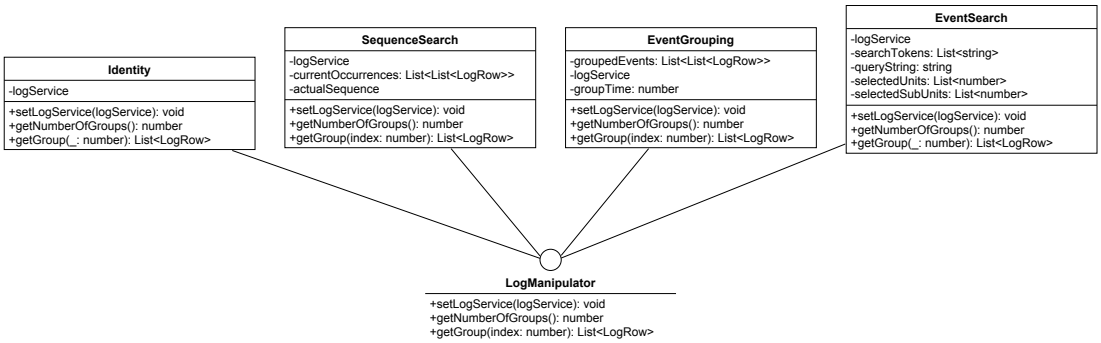

Figura 4.4: Diagramma delle classi del LogManipulator.

– **LogManipulator**
Interfaccia che definisce come presentare i dati alla tabella e al grafico.

Metodi pubblici:

+ `setLogService(logService: LogService): void`
Imposta il logservice che fornisce i dati;

+ `getNumberOfGroups(): number`
Specifica quanti gruppi di risultati sono stati generati;

+ `getGroup(index: number): List<LogRow>`
Ottiene uno dei gruppi di risultati.

- **Identity**
  Questa classe si occupa di fornire al grafico e alla tabella gli eventi del logservice.

  Attributi privati:

  - `logService: LogService`
    Log service che fornisce gli eventi da mostrare successivamente alla view;

  Metodi pubblici:

  + `setLogService(logService: LogService): void`
    Imposta il logservice di cui rilanciare gli eventi;
  + `getNumberOfGroups(): number`
    Ottiene sempre 1 perché questa funzionalità non prevede il raggruppamento;
  + `getGroup(_:number): List<LogRow>`
    Ritorna tutta la lista degli eventi forniti dal logservice.

  Interfacce implementate:

  * `LogManipulator`
    Interfaccia che definisce come presentare i dati alla tabella e al grafico.
- **SequenceSearch**
  Questa classe si occupa di cercare una sequenza di eventi specificata.

  Attributi privati:

  - `logService: LogService`
    Log service che fornisce gli eventi;
  - `currentOccurrences: List<List<LogRow>>`
    Le occorrenze derivanti dalla ricerca di una sequenza.
  - `actualSequence: Sequence`
    Descrizione.

  Metodi pubblici:

  + `setLogService(logService: LogService): void`
    Imposto il log service ed effettuo la ricerca;
  + `getNumberOfGroups(): number`
    Ottiene il numero di risultati da mostrare in gruppo;
  + `getGroup(index:number): List<LogRow>`
    Ottiene un risultato di ricerca.

  Interfacce implementate:

  * `LogManipulator`
    Interfaccia che definisce come presentare i dati alla tabella e al grafico.
- **EventGrouping**
  Questa classe si occupa di gestire il ragguppamento degli eventi su base temporale.

  Attributi privati:

- `groupedEvents: List<List<LogRow>>`
  Insieme dei gruppi di eventi calcolati;
- `logService : LogService`
  Il logservice per ottenere la lista di eventi, può essere null perchè non viene assegnato dal costruttore;
- `groupTime: number`
  La durata temporale di un gruppo di eventi.

Metodi pubblici:

+ `setLogService(logService: LogService): void`
  Imposta il logService e calcola tutti i gruppi di eventi;
+ `getNumberOfGroups(): number`
  Ottiene il numero di gruppi di eventi calcolati;
+ `getGroup(index: number): List<LogRow>`
  Ottiene il gruppo di eventi identificati dall'indice.

Interfacce implementate:

* `LogManipulator`
  Interfaccia che definisce come presentare i dati alla tabella e al grafico.

– **EventSearch**
Questa classe si occupa di filtrare gli eventi in base ad una query di ricerca.

Attributi privati:

- `logService: LogService`
  Log service che fornisce l'elenco di eventi;
- `searchTokens: List<string>`
  Lista dei token di ricerca;
- `queryString: string`
  Query di ricerca;
- `selectedUnits: List<number>`
  Lista delle Unit selezionate;
- `selectedSubUnits: List<number>`
  Lista delle SubUnit selezionate.

Metodi pubblici:

+ `setLogService(logService: LogService): void`
  Imposta il logService su cui eseguire le ricerche;
+ `getNumberOfGroups(): number`
  Ottiene sempre 1 perchè la ricerca non produce gruppi multipli;
+ `getGroup(_: number): List<LogRow>`
  Ottene i risultati della ricerca.

Interfacce implementate:

* `LogManipulator`
  Interfaccia che definisce come presentare i dati alla tabella e al grafico.

4.1.3.3 Barra di navigazione

La barra di navigazione è l'elemento utilizzato per muoversi all'interno dell'applicazione; viene condivisa da tutte le schermate.
Nella Schermata caricamento file che si vede all'avvio dell'applicazione, la barra di navigazione mostra solamente il nome dell'applicazione e un link alla schermata di caricamento file; nel momento in cui ci sarà un file valido caricato, la barra di navigazione mostrerà anche i link alle schermate di visualizzazione dei dati in forma tabellare e in forma grafica.

- **Percorsi:**
  - `/file-upload`;
  - `/table`;
  - `/chart`.
- **Elementi:**
  - Nome applicazione;
  - Link alla schermata di caricamento file;
  - Link alla schermata di visualizzazione tabellare dei dati;
  - Link alla schermata di visualizzazione grafica dei dati.
- **Architettura:**

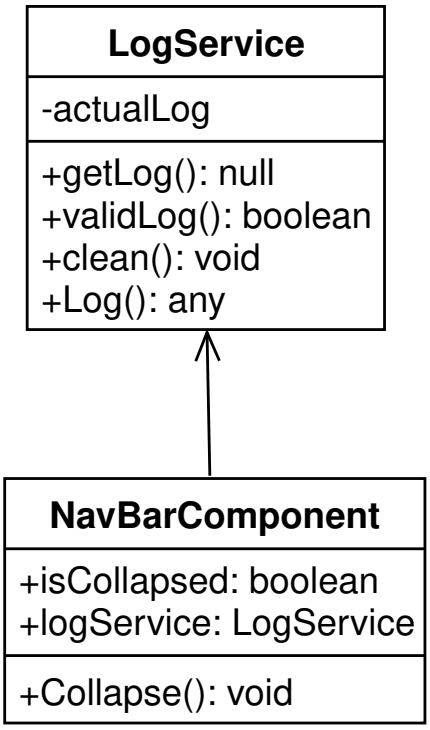

Figura 4.5: Diagramma delle classi del NavBarComponent.

  - **NavBarComponent**
    Classe che gestisce il comportamento della navbar.

Annotazioni:

* `@Component`
  Viene definita la configurazione della classe.

Attributi pubblici:

+ `isCollapsed: boolean`
  Indica quando la navbar è chiusa, è necessario per gestire il layout per schermi piccoli;
+ `logService: LogService`
  LogService utilizzato per sapere quando ci sono degli eventi caricati ed è possibile attivare la visualizzazione a tabella o grafico.

Metodi pubblici:

+ `Collapse: void`
  Metodo che si occupa di gestire l'apertura e la chiusura della navbar alla pressione del pulsante.

– LogService.

4.1.3.4 Schermata caricamento file

La schermata di caricamento file è la schermata che si vede all'avvio dell'applicazione; quando non è ancora stato caricato alcun file valido si vedono solamente il riquadro per il caricamento del file e il pulsante di caricamento.
Durante il caricamento, viene mostrata una barra che indica la percentuale di completamento del caricamento del file.
Una volta caricato un file valido, la schermata mostra anche il nome del file caricato.

- **Percorso:**
  - `/file-upload`.
- **Elementi:**
  - Riquadro per il caricamento del file, sia tramite drag and drop che tramite pulsante;
  - Pulsante per il caricamento del file;
  - Barra con la percentuale di completamento del caricamento del file;
  - Riquadro per la visualizzazione del nome del file caricato.
- **Requisiti soddisfatti:**
  - **RFV-1.1**: L'utente vuole caricare un file di log;
  - **RFV-1.1.1**: L'utente visualizza un messaggio positivo in caso di successo nel caricamento di un file di log;
  - **RFV-1.2**: L'utente ha provato a caricare un file impossibile da leggere correttamente e quindi visualizza un messaggio di errore;
  - **RFV-1.2.1**: L'utente visualizza un errore dato dal formato non corretto di almeno una linea del file;

– **RFV-1.2.2**: L'utente visualizza un errore generato dai dati del file che non si presentano nel formato corretto.

- **Architettura:**

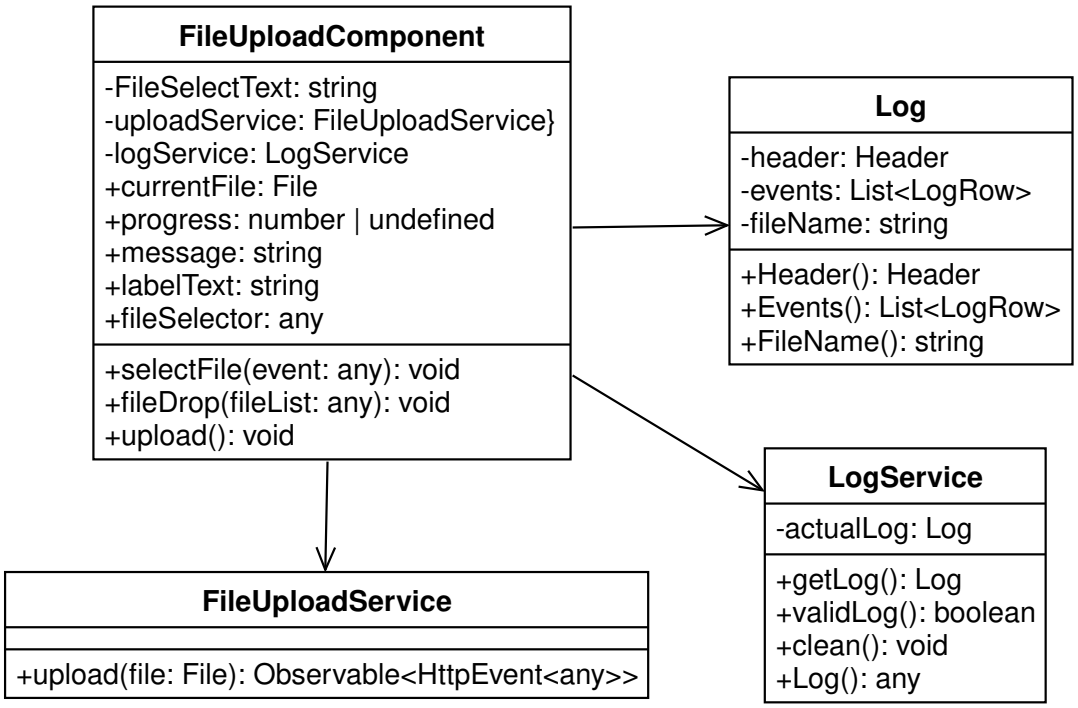

Figura 4.6: Diagramma delle classi del FileUploadComponent.

– **FileUploadComponent**
Classe che definisce il comportamento del widget di caricamento dei file.

Annotazioni:

* `@Component`
Viene definita la configurazione della classe.

Attributi privati:

– `FileSelectText: string`
Scritta di default per la selezione del file;

– `uploadService: FileUploadService`
Service che si occupa di caricare sul server i file;

– `logService: LogService`
Service che si occupa di passare agli altri widget il file di log ricevuto dal server.

Attributi pubblici:

+ `currentFile: File`
Il file attualmente selezionato;

+ `progress: number | undefined`
Il progresso auttuale di caricamento, se è undefined fa sparire dalla view la progress bar;

+ `message: string`
  Messaggio di errore;
+ `@ViewChild("fileSelector") fileSelector: any;`
  Gestore del controllo di input;
+ `labelText: string`
  Il testo della label di selezione del file.

Metodi pubblici:

+ `selectFile(event:any): void`
  Gestisce la selezione dei file tramite dialog, prendendo solo il primo file della lista ed ignorando tutti gli altri;
+ `fileDrop(fileList:any): void`
  Gestisce il drag and drop del file, prendendo solo il primo file della lista se è un csv e ignorando tutti gli altri;
+ `upload(): void`
  Avvia il processo di carcamento del file di log.

– **FileUploadService**
Questo service si occupa di effettuare l'upload del file di log e riceverne i dati.

Annotazioni:

* `@Injectable`
  Per poter essere identificato all'interno del sistema di dependency injection di Angular.

Metodi pubblici:

+ `upload(file:File): Observable<HttpEvent<any>>`
  Effettua la chiamata al server.

– LogService;
– Log.

4.1.3.5 Informazioni sul file caricato

Le informazioni sul file caricato sono le informazioni che vengono mostrate nelle schermate di visualizzazione dei dati in forma tabellare e in forma grafica.
Le informazioni sono riassunte in una tabella in cui la parte superiore riporta i valori di PCDate e UPSDate del file caricato, mentre la parte inferiore mostra i FileName e le rispettive Unit e SubUnit.

- **Percorsi:**
  – `/table`;
  – `/chart`.
- **Elementi:**
  – Riga della tabella contenenente la *PCDate*;
  – Riga della tabella contenenente la *UPSDate*;
  – Tabella contenenente i *FileName* e le rispettive *Unit* e *SubUnit*.

- **Architettura:**

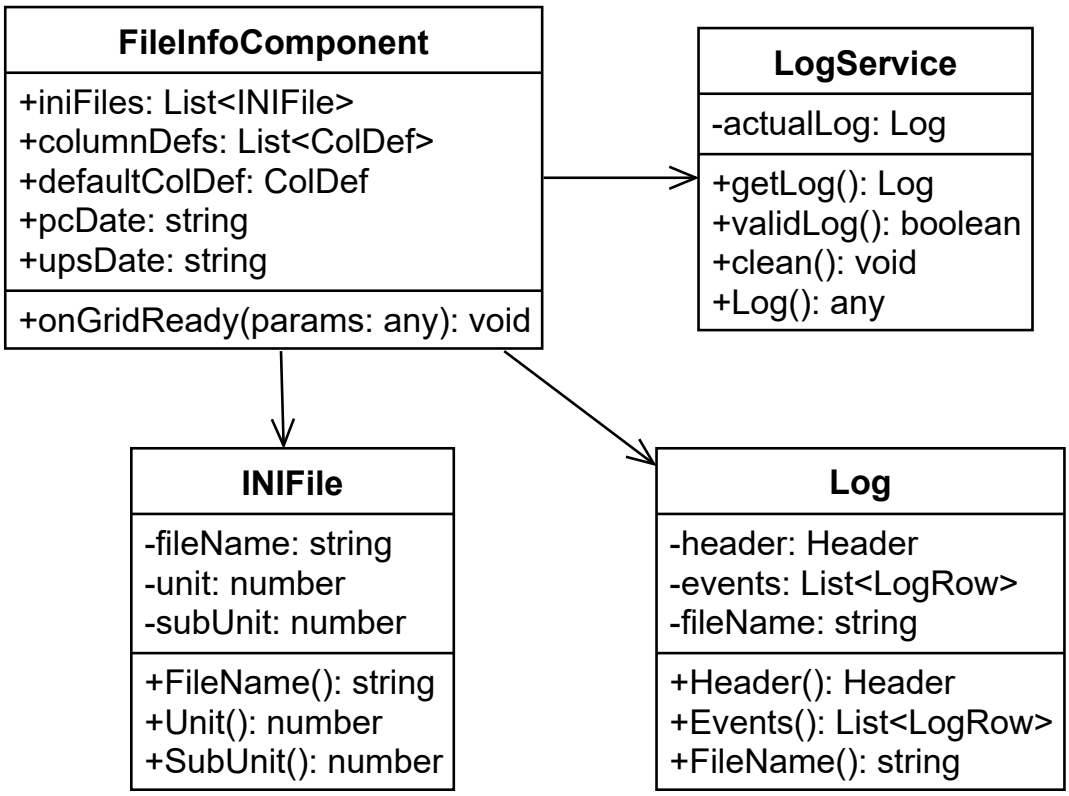

Figura 4.7: Diagramma delle classi del FileInfoComponent.

– **FileInfoComponent**
Questa classe fornisce il comportamento per il widget che mostra le informazioni del log.

Annotazioni:

* `@Component`
Viene definita la configurazione della classe.

Attributi pubblici:

+ `iniFiles: List<INIFile>`
Lista degli INI File del log;
+ `columnDefs: List<ColDef>`
Descrizione delle colonne della tabella degli INI File;
+ `defaultColDef: ColDef`
Impostazioni di default della tabella;
+ `pcDate: string`
Data del pc del momento in cui il log è stato scaricato;
+ `upsDate: string`
Data dell'ups del momento in cui il log è stato scaricato.

Metodi pubblici:

+ `onGridReady(params:any): void`
  Metodo che gestisce l'evento gridReady della tabella.

– INIFile;
– LogService;
– Log.

4.1.3.6 Ricerca di sequenze note

Nei file di log sono presenti alcune sequenze note di eventi che potrebbero essere di particolare interesse per l'utente.
La ricerca di sequenze note permette di cercare una sequenza di eventi nota all'interno del file di log caricato selezionando il nome della sequenza dal menù.

- **Percorso:**
  - `/table`.
- **Elementi:**
  - Casella con scelta a tendina delle sequenze da ricercare;
  - Pulsante di ricerca.
- **Requisiti soddisfatti:**
  - **RFV-1.14**: L'utente vuole individuare le occorrenze di un determinato insieme di eventi (routine di macchinario);
  - **RFV-1.14.1**: L'utente vuole individuare le occorrenze di un determinato insieme di eventi (routine di macchinario) presenti in un determinato ordine.
- **Architettura:**

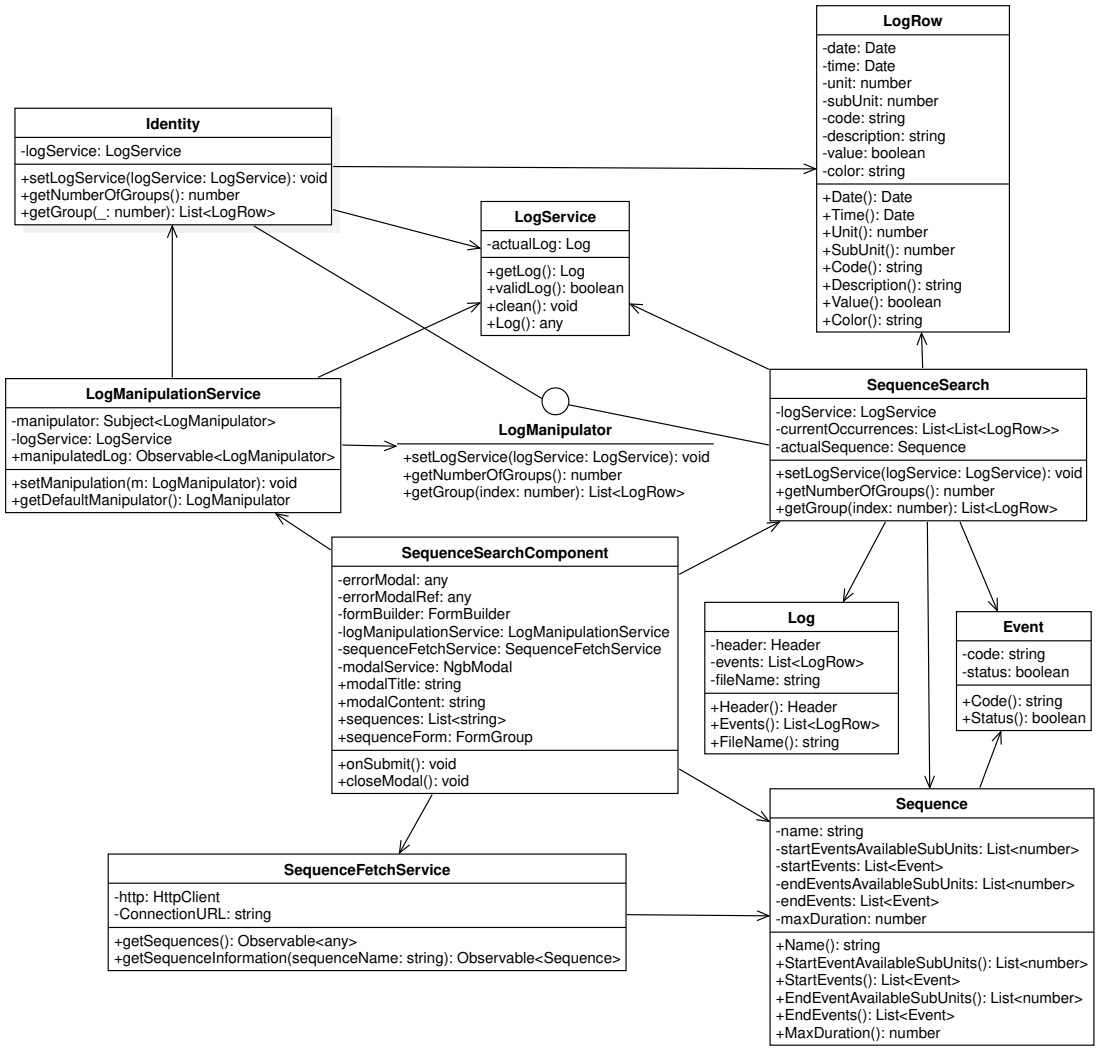

Figura 4.8: Diagramma delle classi del SequenceSearchComponent.

– **SequenceSearchComponent**
Questa classe si occupa di cercare una sequenza di eventi specificata.

Annotazioni:

* `@Component`
Viene definita la configurazione della classe.

Attributi privati:

– `@ViewChild('errorModal') errorModal: any`
Collegamento al template del dialog di errore;
– `errorModalRef: any`
Riferimento al dialog aperto;
– `formBuilder: FormBuilder`
Formbuilder;
– `logManipulationService: LogManipulationService`
Servizio che comunica ai controlli di visualizzazione del log il nuovo LogManipulator;

- `sequenceFetchService: SequenceFetchService`
  Servizio che si occupa di ottenere le informazioni sulle sequenze;
- `modalService: NgbModal`
  Servizio che si occupa di gestire i modal di bootstrap.

Attributi pubblici:

+ `modalTitle: string`
  Titolo del dialog che mostra un errore di comunucazione con il backend;
+ `modalContent: string`
  Contenuto del dialog che mostra un errore di comunicazione con il backend;
+ `sequences: List<string>`
  Elenco dei nomi delle sequenze note disponibili;
+ `sequenceForm: FormGroup`
  Controllo che ragguppa gli input del form.

Metodi pubblici:

+ `onSubmit(): void`
  Metodo che gestisce il submit del form;
+ `closeModal(): void`
  Metodo che gestisce la chiusura del dialog di errore.

– **Sequence**
Questa classe rappresenta una sequenza nota.

Attributi privati:

- `name: string`
  Nome della sequenza nota;
- `startEventsAvailableSubUnits: List<number>`
  Lista delle subunit sulle quale possono scatenarsi gli eventi di inizio della sequenza;
- `startEvents: List<Event>`
  Lista degli eventi che identificano l'inizio della sequenza;
- `endEventsAvailableSubUnits: List<number>`
  Lista delle subunit sulle quali possono scatenarsi gli eventi di fine della sequenza;
- `endEvents: List<Event>`
  Lista degli eventi che identificano l'inizio della sequenza;
- `maxDuration: number`
  Durata massima della sequenza in millisecondi.

Metodi pubblici:

+ `Name(): string`
  Ottene il nome della sequenza;
+ `StartEventAvailableSubUnits(): List<number>`
  Ottene la lista delle sub unit sulle quali possono scatenarsi gli eventi di inizio sequenza;
+ `StartEvents(): List<Event>`
  Ottene la lista di eventi che identificano l'inizio della sequenza;

+ `EndEventAvailableSubUnits(): List<number>`
  Ottene la lista delle subunit sulle quali possono scatenarsi gli eventi di fine sequenza;
+ `EndEvents(): List<Event>`
  Ottene la lista di eventi che identificano la fine della sequenza;
+ `MaxDuration(): number`
  Ottene la durata massima della sequenza in millisecondi.

– **SequenceSearch**
Questa classe si occupa di cercare una sequenza di eventi specificata.

Attributi privati:

– `logService: LogService`
  Log service che fornisce gli eventi;
– `currentOccurrences: List<List<LogRow>>`
  Le occorrenze derivanti dalla ricerca di una sequenza.
– `actualSequence: Sequence`
  Descrizione.

Metodi pubblici:

+ `setLogService(logService: LogService): void`
  Imposto il log service ed effettuo la ricerca;
+ `getNumberOfGroups(): number`
  Ottiene il numero di risultati da mostrare in gruppo;
+ `getGroup(index:number): List<LogRow>`
  Ottiene un risultato di ricerca.

Interfacce implementate:

∗ `LogManipulator`
  Interfaccia che definisce come presentare i dati alla tabella e al grafico.

– **Event**
Questa classe rappresenta un'occorrenza di un evento che deve essere trovata per identificare una fase della sequenza.

Attributi privati:

– `code: string`
  Code dell'evento;
– `status: boolean`
  Valore preso dall'evento .

Metodi pubblici:

+ `Code(): string`
  Ottiene il code dell'evento;
+ `Status() : boolean`
  Ottiene il valore che prende l'evento.

– **LogManipulationService**
Questo service si occupa di trasmettere alla tabella e al grafico l'oggetto che

filtra i dati del log da visualizzare.

Annotazioni:

* `@Injectable`
  Per poter essere identificato all'interno del sistema di dependency injection di Angular.

Attributi privati:

- `manipulator : Subject<LogManipulator>`
  Subject che prende i nuovi log manipulator per notificarli alla view;
- `logService: LogService`
  Il log service con tutti gli eventi.

Attributi pubblici:

+ `manipulatedLog : Observable<LogManipulator>`
  Observable al quale la view effettuerà la subscribe per ottenere i nuovi log manipulator.

Metodi pubblici:

+ `setManipulation(m: LogManipulator): void`
  Imposta il nuovo LogManipulator al quale verrà assegnato il LogService prima di essere mandato alla view;
+ `getDefaultManipulator(): LogManipulator`
  Ottiene il logManipulator Identity, considerato di default.

– **SequenceFetchService**
Questo service si occupa di comunicare con il backend le informazioni relative alle sequenze note.

Annotazioni:

* `@Injectable`
  Per poter essere identificato all'interno del sistema di dependency injection di Angular.

Attributi privati:

- `http: HttpClient`
  Il client http che si occuperà di comunicare con il server;
- `ConnectionURL: string`
  URL del backend.

Metodi pubblici:

+ `getSequences(): Observable<any>`
  Ottiene la lista di nomi delle sequenze;
+ `getSequenceInformation(sequenceName: string): Observable<Sequence>`
  Ottiene tutte le informazioni relative alla sequenza specificata.

– Log;
– LogService;

- LogRow;
- LogManipulator;
- Identity;

4.1.3.7 Raggruppamento degli eventi

Il raggruppamento degli eventi permette di raggruppare gli eventi in base ad un valore temporale inserito dall'utente e di visualizzare gli eventi raggruppati nella tabella; per raggruppare gli eventi sarà chiesto di inserire un valore numerico e di scegliere un'unità di misura tra quelle proposte dal menù a tendina (di default il raggruppamento viene effettuato in ms).

- **Percorso:**
  - `/table`.
- **Elementi:**
  - Casella per l'inserimento del valore numerico per cui raggruppare;
  - Casella con scelta a tendina dell'unità di misura;
  - Pulsante di raggruppamento.
- **Requisiti soddisfatti:**
  - **RFV-1.5**: L'utente vuole visualizzare gli eventi raggruppati per una durata temporale scelta;
  - **RFV-1.6**: L'utente vuole selezionare una durata temporale.
- **Architettura:**

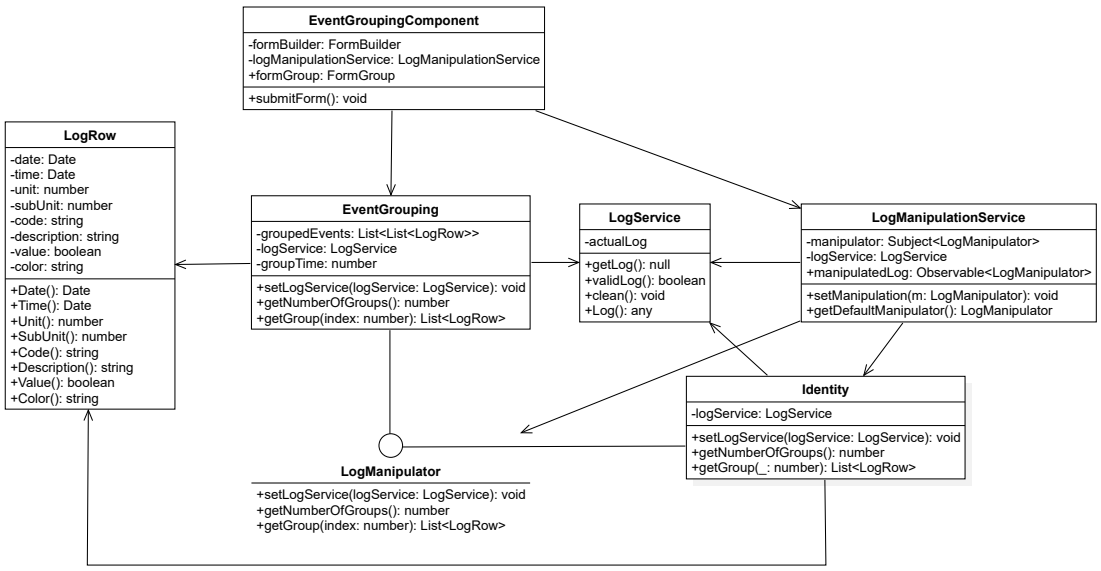

Figura 4.9: Diagramma delle classi del EventGroupingComponent.

– **EventGroupingComponent**
Controller che gestisce il widget di raggruppamento degli eventi.

Annotazioni:

* `@Component`
Viene definita la configurazione della classe.

Attributi privati:

– `formBuilder: FormBuilder`
Servizio di gestione dei form;
– `logManipulationService: LogManipulationService`
Servizio di gestione dei nuovi logManipulator.

Attributi pubblici:

+ `formGroup: FormGroup`
Gestore del form.

Metodi pubblici:

+ `submitForm(): void`
Metodo che gestisce il submit del form, comunica la log manipulation service il nuovo log manipulator. Se il numero di gruppi specificato è 0 verrà impostato il manipulator di default.

– **EventGrouping**
Questa classe si occupa di gestire il ragguppamento degli eventi su base temporale.

Attributi privati:

– `groupedEvents: List<List<LogRow>>`
Insieme dei gruppi di eventi calcolati;
– `logService : LogService`
Il logservice per ottenere la lista di eventi, può essere null perchè non viene assegnato dal costruttore;
– `groupTime: number`
La durata temporale di un gruppo di eventi.

Metodi pubblici:

+ `setLogService(logService: LogService): void`
Imposta il logService e calcola tutti i gruppi di eventi;
+ `getNumberOfGroups(): number`
Ottiene il numero di gruppi di eventi calcolati;
+ `getGroup(index: number): List<LogRow>`
Ottiene il gruppo di eventi identificati dall'indice.

Interfacce implementate:

* `LogManipulator`
Interfaccia che definisce come presentare i dati alla tabella e al grafico.

– **LogManipulationService**
Questo service si occupa di trasmettere alla tabella e al grafico l'oggetto che

filtra i dati del log da visualizzare.

Annotazioni:

* `@Injectable`
  Per poter essere identificato all'interno del sistema di dependency injection di Angular.

Attributi privati:

- `manipulator : Subject<LogManipulator>`
  Subject che prende i nuovi log manipulator per notificarli alla view;
- `logService: LogService`
  Il log service con tutti gli eventi.

Attributi pubblici:

+ `manipulatedLog : Observable<LogManipulator>`
  Observable al quale la view effettuerà la subscribe per ottenere i nuovi log manipulator.

Metodi pubblici:

+ `setManipulation(m: LogManipulator): void`
  Imposta il nuovo LogManipulator al quale verrà assegnato il LogService prima di essere mandato alla view;
+ `getDefaultManipulator(): LogManipulator`
  Ottiene il logManipulator Identity, considerato di default.

– LogManipulator;
– LogService;
– LogRow;
– Identity.

4.1.3.8 Operazioni nella tabella

Nella tabella è possibile selezionare una delle operazioni proposte nel menù a fisarmonica ovvero la ricerca di sequenze note oppure il raggruppamento degli eventi.

- **Percorso:**
  - `/table`.
- **Elementi:**
  - Campo per la ricerca di sequenze note;
  - Campo per il raggruppamento degli eventi.
- **Architettura:**

Figura 4.10: Diagramma delle classi del TableSearchComponent.

– **TableSearchComponent**
Classe che definisce il comportamento del widget che aggrega i controlli di manipolazione dei record del log per la tabella.

Annotazioni:

* `@Component`
Viene definita la configurazione della classe.

4.1.3.9 Intestazione della tabella

Sopra la tabella è presente un menù a fisarmonica che contiene le informazioni sul file caricato e le possibili operazioni da fare nella tabella.

- **Percorso:**
  - `/table`.
- **Elementi:**
  - Informazioni sul file caricato;
  - Operazioni nella tabella.
- **Architettura:**

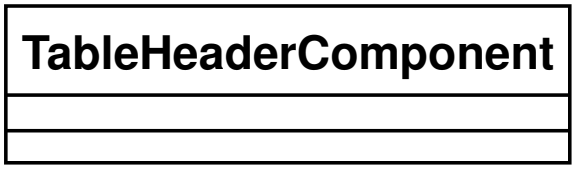

Figura 4.11: Diagramma delle classi del TableHeaderComponent.

– **TableHeaderComponent**
Classe che definisce il comporamento del widget di header della tabella.

Annotazioni:

* `@Component`
Viene definita la configurazione della classe.

4.1.3.10 Tabella

La pagina della tabella contiene l'intestazione della tabella, un campo in cui inserire il numero della tabella da visualizzare, che viene visualizzato solamente se si sono ricercate sequenze note o raggruppamenti, una tabella contenente la data del primo evento visualizzato, la data dell'ultimo evento visualizzato e il numero di eventi visualizzati e la tabella degli eventi in cui sono visualizzati gli eventi del file di log caricato e su cui si possono effettuare le operazioni di ordinamento e filtraggio per colonne.

- **Percorso:**
  - `/table`.
- **Elementi:**
  - Intestazione della tabella;
  - Campo in cui inserire il numero della tabella da visualizzare;
  - Data del primo evento visualizzato;
  - Data dell'ultimo evento visualizzato;
  - Numero di eventi visualizzati;
  - Tabella degli eventi.
- **Requisiti soddisfatti:**
  - **RFV-1.3**: L'utente visualizza gli eventi del file di log;
  - **RFV-1.3.1**: L'utente visualizza gli eventi del file di log in forma tabellare;
  - **RFV-1.4**: L'utente vuole poter ordinare la tabella rispetto ad un campo dati;
  - **RFV-1.4.1**: L'utente vuole poter ordinare i dati nella tabella secondo la colonna Data/Ora;
  - **RFV-1.4.2**: L'utente vuole poter ordinare i dati nella tabella secondo la colonna Code;
  - **RFV-1.4.3**: L'utente vuole poter ordinare i dati nella tabella secondo la colonna Unit;
  - **RFV-1.4.4**: L'utente vuole poter ordinare i dati nella tabella secondo la colonna SubUnit;
  - **RFV-1.11**: L'utente vuole applicare un filtro sui dati che sta visualizzando;
  - **RFV-3.10**: L'utente vuole poter visualizzare delle informazioni aggiuntive su un evento sul grafico;
  - **RFV-1.11.2**: L'utente vuole filtrare i dati del log in base al valore del campo Unit, visualizzando quelli che lo rispettano;
  - **RFV-1.11.3**: L'utente vuole filtrare i dati del log in base al valore del campo SubUnit, visualizzando quelli che lo rispettano;
  - **RFV-1.11.4**: L'utente vuole filtrare i dati del log in base al valore del campo Code, visualizzando quelli che lo rispettano;

– **RFV-1.16**: L'utente visualizza un avviso dovuto all'assenza di dati da visualizzare.

- **Architettura:**

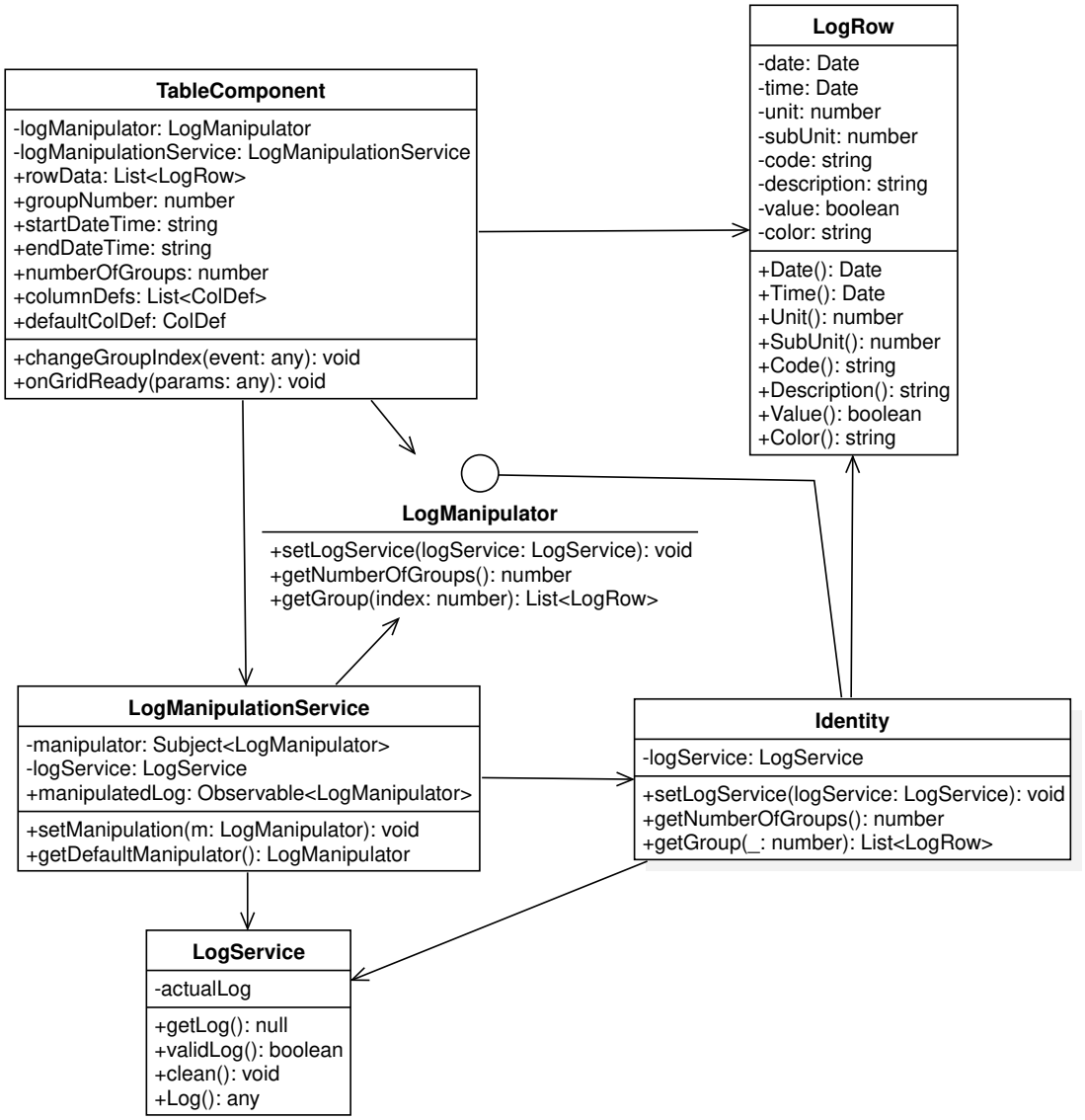

Figura 4.12: Diagramma delle classi del TableComponent.

– **TableComponent**
Classe che definisce il comportamento della tabella di visualizzazione degli eventi.

Annotazioni:

∗ `@Component`
Viene definita la configurazione della classe.

Attributi privati:

– `logManipulator: LogManipulator`
LogManipulator che fornisce i dati da mostrare;

  - `logManipulationService: LogManipulationService`
    Il log manipulator che fornisce gli eventi da mostrare.

Attributi pubblici:

  + `rowData: List<LogRow>`
    Elenco dei dati da mostrare sulla tabella;
  + `groupNumber: number`
    Indice del gruppo di dati da visualizzare;
  + `numberOfGroups: number`
    Numero totale di gruppi di dati visualizzabili;
  + `startDateTime: string`
    Data e ora del primo evento nella tabella;
  + `endDateTime: string`
    Data e ora dell'ultimo evento nella tabella;
  + `columnDefs: List<ColDef>`
    Definizione dei campi dati da mostrare nella tabella;
  + `defaultColDef: ColDef`
    Impostazione di default dei campi della tabella.

Metodi pubblici:

  + `changeGroupIndex(event: any): void`
    Metodo che gestisce il cambio del numero di indice del gruppo da visualizzare dalla view;
  + `onGridReady(params: any): void`
    Metodo che gestisce l'evento gridReady emesso dalla tabella.

- **LogManipulationService**
  Questo service si occupa di trasmettere alla tabella e al grafico l'oggetto che filtra i dati del log da visualizzare.

  Annotazioni:

  * `@Injectable`
    Per poter essere identificato all'interno del sistema di dependency injection di Angular.

  Attributi privati:

  - `manipulator : Subject<LogManipulator>`
    Subject che prende i nuovi log manipulator per notificarli alla view;
  - `logService: LogService`
    Il log service con tutti gli eventi.

  Attributi pubblici:

  + `manipulatedLog : Observable<LogManipulator>`
    Observable al quale la view effettuerà la subscribe per ottenere i nuovi log manipulator.

  Metodi pubblici:

  + `setManipulation(m: LogManipulator): void`
    Imposta il nuovo LogManipulator al quale verrà assegnato il LogService prima di essere mandato alla view;

+ `getDefaultManipulator(): LogManipulator`
  Ottiene il logManipulator Identity, considerato di default.

– LogManipulator;
– LogRow;
– LogService;
– Identity.

4.1.3.11 Ricerca di eventi specifici

La ricerca di eventi specifici permette di ricercare gli eventi all'interno del file di log caricato inserendo una stringa di ricerca e selezionando le Unit e le SubUnit da considerare nella ricerca.

- **Percorso:**
  - `/chart`.
- **Elementi:**
  - Casella di testo in cui inserire la stringa di ricerca;
  - Menù a tendina in cui scegliere le Unit da considerare nella ricerca tra quelle presenti nel file di log;
  - Menù a tendina in cui scegliere le SubUnit da considerare nella ricerca tra quelle presenti nel file di log;
  - Pulsante di ricerca.
- **Requisiti soddisfatti:**
  - **RFV-1.11.2**: L'utente vuole filtrare i dati del log in base al valore del campo Unit, visualizzando quelli che lo rispettano;
  - **RFV-1.11.3**: L'utente vuole filtrare i dati del log in base al valore del campo SubUnit, visualizzando quelli che lo rispettano;
  - **RFV-1.11.4**: L'utente vuole filtrare i dati del log in base al valore del campo Code, visualizzando quelli che lo rispettano;
  - **RFV-1.13**: L'utente vuole cercare un insieme di eventi che abbiano, tra i loro dati, una stringa di testo inserita dall'utente.
- **Architettura:**

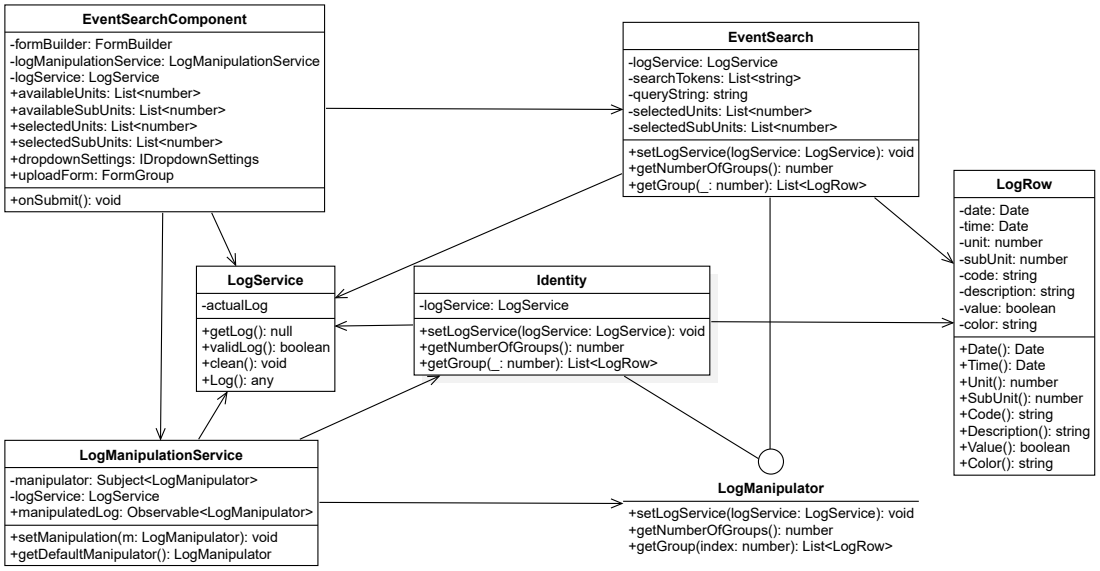

Figura 4.13: Diagramma delle classi del EventSearchComponent.

– **EventSearchComponent**
Questa classe definisce il controller del widget event-search.

Annotazioni:

∗ `@Component`
Viene definita la configurazione della classe.

Attributi privati:

– `formBuilder: FormBuilder`
Servizio che gestisce i form;
– `logManipulationService: LogManipulationService`
Servizio che segnala alla tabella e al grafico la presenza di un nuovo logManipulator;
– `logService: LogService`
Il logService che contiene tutti gli eventi del log.

Attributi pubblici:

\+ `availableUnits: List<number>`
Lista delle Unit disponibili;
\+ `availableSubUnits: List<number>`
Lista delle SubUnit disponibili;
\+ `selectedUnits: List<number>`
Lista delle Unit selezionate;
\+ `selectedSubUnits: List<number>`
Lista delle SubUnit selezionate;
\+ `dropdownSettings: IDropdownSettings`
Impostazioni per la selezione nei menù a tendina;
\+ `uploadForm: FormGroup`
Gestore della form.

Metodi pubblici:

+ `onSubmit(): void`
  Metodo che gestisce il submit del form, se la query string è vuota si manda il manipulator di default.

– **EventSearch**
Questa classe si occupa di filtrare gli eventi in base ad una query di ricerca.

Attributi privati:

– `logService: LogService`
  Log service che fornisce l'elenco di eventi;
– `searchTokens: List<string>`
  Lista dei token di ricerca;
– `queryString: string`
  Query di ricerca;
– `selectedUnits: List<number>`
  Lista delle Unit selezionate;
– `selectedSubUnits: List<number>`
  Lista delle SubUnit selezionate.

Metodi pubblici:

+ `setLogService(logService: LogService): void`
  Imposta il logService su cui eseguire le ricerche;
+ `getNumberOfGroups(): number`
  Ottiene sempre 1 perchè la ricerca non produce gruppi multipli;
+ `getGroup(_: number): List<LogRow>`
  Ottene i risultati della ricerca.

Interfacce implementate:

∗ `LogManipulator`
  Interfaccia che definisce come presentare i dati alla tabella e al grafico.

– **LogManipulationService**
Questo service si occupa di trasmettere alla tabella e al grafico l'oggetto che filtra i dati del log da visualizzare.

Annotazioni:

∗ `@Injectable`
  Per poter essere identificato all'interno del sistema di dependency injection di Angular.

Attributi privati:

– `manipulator : Subject<LogManipulator>`
  Subject che prende i nuovi log manipulator per notificarli alla view;
– `logService: LogService`
  Il log service con tutti gli eventi.

Attributi pubblici:

+ `manipulatedLog : Observable<LogManipulator>`
  Observable al quale la view effettuerà la subscribe per ottenere i nuovi log manipulator.

Metodi pubblici:

+ `setManipulation(m: LogManipulator): void`
  Imposta il nuovo LogManipulator al quale verrà assegnato il LogService prima di essere mandato alla view;
+ `getDefaultManipulator(): LogManipulator`
  Ottiene il logManipulator Identity, considerato di default.

– LogManipulator;
– LogService;
– Identity;
– LogRow.

4.1.3.12 Operazioni nel grafico

Nel grafico è possibile selezionare l'operazione proposta nel menù a fisarmonica ovvero la ricerca di eventi specifici.

- **Percorso:**
  – `/chart`.
- **Elementi:**
  – Campo per la ricerca di eventi specifici.
- **Architettura:**

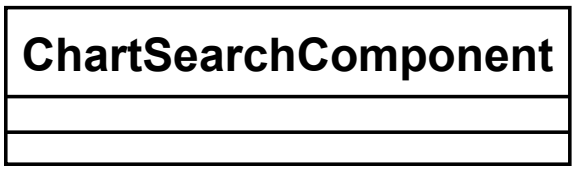

Figura 4.14: Diagramma delle classi del ChartSearchComponent.

– **ChartSearchComponent**
Classe che definisce il comportamento del widget che aggrega i controlli di manipolazione dei record del log per il grafico.

Annotazioni:

* `@Component`
  Viene definita la configurazione della classe.

4.1.3.13 Intestazione del grafico

Sopra al grafico è presente un menù a fisarmonica che contiene le informazioni sul file caricato e le possibili operazioni da fare nel grafico.

- **Percorso:**
  - `/chart`.
- **Elementi:**
  - Informazioni sul file caricato;
  - Operazioni nel grafico.
- **Architettura:**

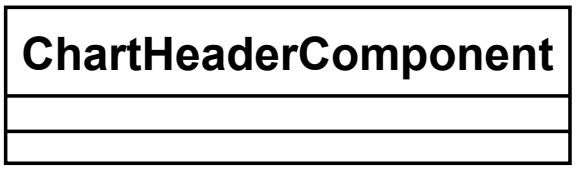

Figura 4.15: Diagramma delle classi del ChartHeaderComponent.

  - **ChartHeaderComponent**
    Classe che definisce il comporamento del widget di header del grafico.

    Annotazioni:
    * `@Component`
      Viene definita la configurazione della classe.

4.1.3.14 Grafico

La pagina del grfico contiene l'intestazione del grafico e il grafico stesso in cui sono visualizzati gli eventi del file di log caricato.

- **Percorso:**
  - `/chart`.
- **Elementi:**
  - Intestazione del grafico;
  - Grafico degli eventi.
- **Requisiti soddisfatti:**
  - **RFV-1.3**: L'utente visualizza gli eventi del file di log;
  - **RFV-1.3.2**: L'utente visualizza gli eventi del file di log in un Horizon Chart di attivazione degli eventi;

- **RFV-1.8**: L'utente vuole poter aumentare o diminuire la risoluzione del grafico rispetto l'asse temporale;
- **RFV-1.9**: L'utente vuole poter cambiare l'offset temporale dei dati visualizzati dal grafico;
- **RFV-3.10**: L'utente vuole poter visualizzare delle informazioni aggiuntive su un evento sul grafico.

- **Architettura:**

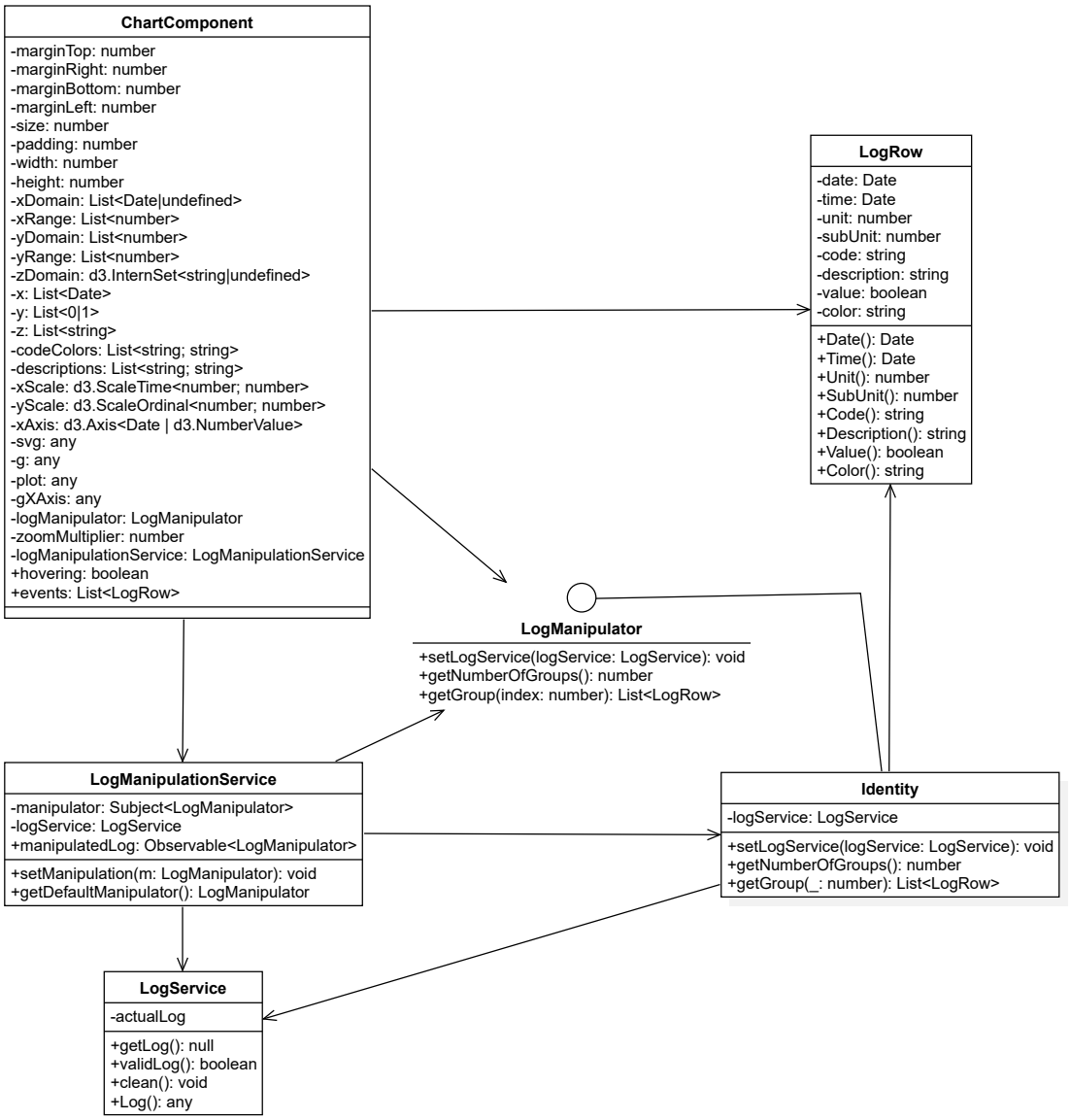

Figura 4.16: Diagramma delle classi del ChartComponent.

- **ChartComponent**
  Classe che definisce il comportamento del grafico per la visualizzazione degli eventi.

  Annotazioni:

* `@Component`
  Viene definita la configurazione della classe.

Attributi privati:

- `marginTop: number`
  Margine superiore del grafico;
- `marginRight: number`
  Margine destro del grafico;
- `marginBottom: number`
  Margine inferiore del grafico;
- `marginLeft: number`
  Margine sinistro del grafico;
- `size: number`
  Altezza della riga del singolo evento;
- `padding: number`
  Padding del grafico;
- `width: number`
  Larghezza del grafico;
- `height: number`
  Altezza del grafico;
- `xDomain: List<Date|undefined>`
  Array contenente le coppie xmin e xmax;
- `xRange: List<number>`
  Range dell'asse x, contiene coppie di valori dei range sinistro e destro;
- `yDomain: List<number>`
  Array contenente le coppie ymin e ymax;
- `yRange: List<number>`
  Range dell'asse y, contiene coppie di valori dei range inferiore e superiore;
- `zDomain: d3.InternSet<string|undefined>`
  Dominio dei valori dell'asse z;
- `x: List<Date>`
  Valori dell'asse x (lista delle date);
- `y: List<0|1>`
  Valori dell'asse y (1 se l'evento è attivo, 0 altrimenti);
- `z: List<string>`
  Valori dell'asse z (lista degli eventi);
- `codeColors: List<string; string>`
  Array con tuple di Code e Colors non ripetute;
- `descriptions: List<string; string>`
  Descrizioni degli eventi;
- `xScale: d3.ScaleTime<number; number>`
  Funzione per convertire una data in una posizione per l'asse x;
- `yScale: d3.ScaleOrdinal<number; number>`
  Funzione per convertire un valore in una posizione per l'asse y;
- `xAxis: d3.Axis<Date | d3.NumberValue>`
  Variabile che contiene le informazioni per disegnare l'asse x;

- `svg: any`
  Variabile che contiene il tag svg;
- `g: any`
  Variabile che contiene i tag g; ogni tag g contiene sia il testo del codice e il suo grafico;
- `plot: any`
  Variabile che contiene ciascun grafico;
- `gXAxis: any`
  Variabile che contiene l'effettivo tag g dell'asse x;
- `logManipulator: LogManipulator`
  Variabile che fornisce i dati da disegnare manipolati in base alle opzioni inserite;
- `zoomMultiplier: number`
  Variabile che identifica il valore massimo a cui si può effettuare lo zoom;
- `logManipulationService: LogManipulationService`
  server che notifica quando vengono aggiornati i parametri di filtraggio.

Attributi pubblici:

+ `hovering: boolean`
  Variabile booleana che contiene che rileva se l'utente sta facendo hover sul grafico (true se sta facendo hover, false altrimenti);
+ `events: List<LogRow>`
  Eventi visualizzati.

– **LogManipulationService**
Questo service si occupa di trasmettere alla tabella e al grafico l'oggetto che filtra i dati del log da visualizzare.

Annotazioni:

* `@Injectable`
  Per poter essere identificato all'interno del sistema di dependency injection di Angular.

Attributi privati:

- `manipulator : Subject<LogManipulator>`
  Subject che prende i nuovi log manipulator per notificarli alla view;
- `logService: LogService`
  Il log service con tutti gli eventi.

Attributi pubblici:

+ `manipulatedLog : Observable<LogManipulator>`
  Observable al quale la view effettuerà la subscribe per ottenere i nuovi log manipulator.

Metodi pubblici:

+ `setManipulation(m: LogManipulator): void`
  Imposta il nuovo LogManipulator al quale verrà assegnato il LogService prima di essere mandato alla view;

+ `getDefaultManipulator(): LogManipulator`
  Ottiene il logManipulator Identity, considerato di default.

– LogManipulator;
– LogService;
– Identity;
– LogRow.

4.1.4 Backend SmartLogStatistics

Le dipendenze del progetto di back-end di `SmartLogStatistics` sono:

- Libreria `Core`

Il progetto di back-end di `SmartLogStatistics` soddisfa i seguenti requisiti:

- **RFS-1.1**: L'utente vuole caricare un insieme di file di log nel database;
- **RFS-1.15**: L'utente vuole raggruppare i dati per uno o più campi;
- **RFS-1.15.1**: L'utente vuole raggruppare i dati in base al valore del campo Code;
- **RFS-1.15.2**: L'utente vuole raggruppare i dati in base al valore del campo data/ora;
- **RFS-1.15.3**: L'utente vuole raggruppare i dati in base al valore della versione del firmware;
- **RFS-1.15.4**: L'utente vuole raggruppare i dati in base al valore del campo Unit;
- **RFS-1.15.5**: L'utente vuole raggruppare i dati in base al valore del campo SubUnit.

I componenti del progetto SmartLogStatistics sono:

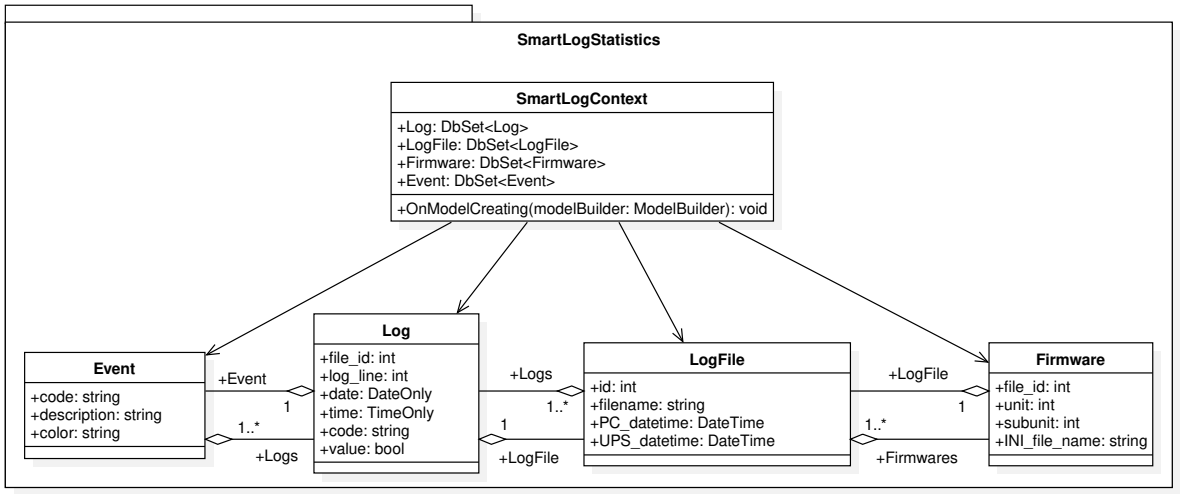

Figura 4.17: Diagramma delle classi di comunicazione col database del progetto SmartLogStatistics.

4.1.4.1 SmartLogContext

- **Metodi**:
  - `OnModelCreating(modelBuilder: ModelBuilder): void`

    Descrizione:
    - Metodo che crea il database nel caso in cui non sia presente o lo sistema nel caso in cui non sia ben costruito.

    Input:
    - `modelBuilder`: oggetto che rappresenta la struttura del modello da creare.
- **Attributi**:
  - `Log`: oggetto di tipo `DbSet<Log>` che rappresenta il contenuto della tabella `Log` nel database;
  - `LogFile`: oggetto di tipo `DbSet<LogFile>` che rappresenta il contenuto della tabella `LogFile` nel database;
  - `Firmware`: oggetto di tipo `DbSet<Firmware>` che rappresenta il contenuto della tabella `Firmware` nel database;
  - `Event`: oggetto di tipo `DbSet<Event>` che rappresenta il contenuto della tabella `Event` nel database.

4.1.4.2 Event

- **Attributi**:
  - `code`: variabile di tipo `string` che rappresenta il codice dell'evento;
  - `description`: variabile di tipo `string` che rappresenta la descrizione dell'evento;
  - `color`: variabile di tipo `string` che rappresenta il colore;
  - `Logs`: Oggetto di tipo `ICollection<Log>` che rappresenta una relazione con la tabella Log.

4.1.4.3 Log

- **Attributi**:
  - `file_id`: variabile di tipo `int` che rappresenta un valore incrementale univoco per ogni file di log;
  - `log_line`: variabile di tipo `int` che rappresenta un valora incrementale univoco per ogni riga di un file di log;
  - `date`: oggetto di tipo `DateOnly` che rappresenta la data in cui si è verificato l'evento;
  - `time`: oggetto di tipo `TimeOnly` che rappresenta l'ora in cui si è verificato l'evento;
  - `code`: variabile di tipo `string` che rappresenta il codice dell'evento;

- `value`: variabile di tipo `bool` che rappresenta il valore dell'evento;
- `Event`: variabile di tipo `Event` che rappresenta una relazione con la tabella Event;
- `LogFile`: variabile di tipo `LogFile` che rappresenta una relazione con la tabella LogFile.

4.1.4.4 LogFile

- **Attributi**:
  - `id`: variabile di tipo `int` che rappresenta un valore incrementale univoco per ogni file di log;
  - `code`: variabile di tipo `string` che rappresenta il codice dell'evento;
  - `PC_date`: oggetto di tipo `DateTime` che rappresenta la data e l'ora in cui è stato ricevuto il file di log;
  - `UPS_datetime`: oggetto di tipo `DateTime` che rappresenta la data e l'ora in cui è stato prodotto il file di log;
  - `Logs`: variabile di tipo `ICollection<Log>` che rappresenta una relazione con la tabella Log;
  - `Firmwares`: variabile di tipo `ICollection<Firmware>` che rappresenta una relazione con la tabella Firmware.

4.1.4.5 Firmware

- **Attributi**:
  - `file_id`: variabile di tipo `int` che rappresenta un valore incrementale univoco per ogni file di log;
  - `unit`: variabile di tipo `int` che rappresenta la unit che contiene la subUnit indicata;
  - `subunit`: variabile di tipo `int` che rappresenta la subunit che utilizza il firmware indicato;
  - `INI_file_name`: variabile di tipo `string` che rappresenta il firmware utilizzato dalla subUnit indicata;
  - `LogFile`: variabile di tipo `LogFile` che rappresenta una relazione con la tabella LogFile.

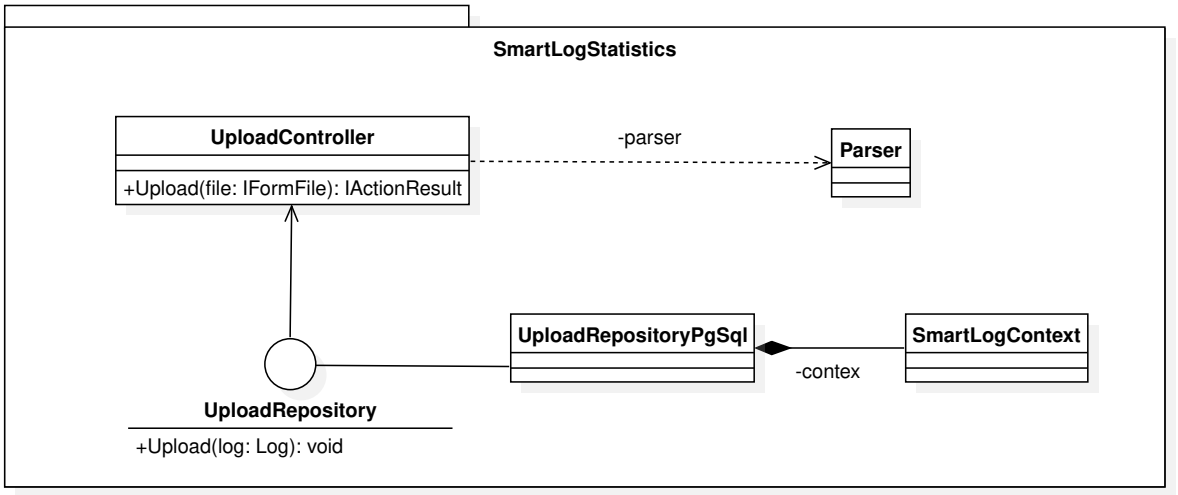

Figura 4.18: Diagramma delle classi per il caricamento dei dati del progetto SmartLog-Statistics.

4.1.4.6 UploadRepository

- **Metodi**:
  - `Upload(log: Log): void`

    Descrizione:
    - Collabora con il context per interagire con il database.

    Input:
    - Oggetto di tipo `Log` che rappresenta il file di log da caricare.

4.1.4.7 UploadRepositoryPgSql

- **Descrizione**:
  - La classe implementa i metodi dell'interfaccia `UploadRepository` tramite una connessione al database effettuata dall'attributo `context`.
- **Attributi**:
  - `context`: oggetto di tipo `SmartLogContext` che rappresenta la connessione con il database.

4.1.4.8 UploadController

- **Metodi**:
  - `Upload(file: IFormFile): IActionResult`

    Descrizione:
    - Gestisce una chiamata *HTTP POST* sull'endpoint `api/upload` e ritorna un oggetto di tipo `IActionResult`.

Input:

* `file`: file ottenuti da una richiesta avente `Content-Type` pari a `multipart/form-data`.

Output:

* Oggetto di tipo `IActionResult` che rappresenta la risposta *HTTP*, la quale può avere esito positivo o negativo (vedi sezione API SmartLogStatistics).

- **Attributi**:
  - `_Parser`: oggetto di tipo `Parser` dedicato al parsing dei file in ingresso.

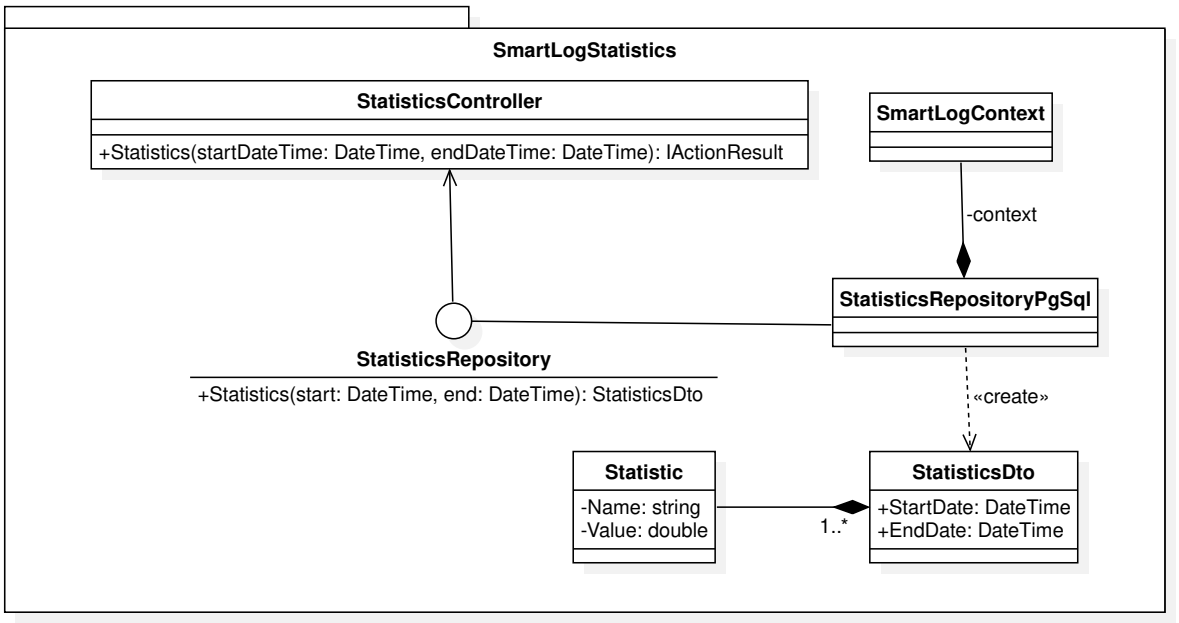

Figura 4.19: Diagramma delle classi per la gestione delle richieste delle statistiche sui file di log del progetto SmartLogStatistics.

4.1.4.9 StatisticsRepository

- **Metodi**:
  - `Statistics(start: DateTime, end: DateTime): StatisticsDto`

    Descrizione:

    * Collabora con il context per interagire con il database. Ritorna un oggetto di tipo `StatisticsDto`.

    Input:

    * `start`: data dell'UPS del primo file di log da prelevare;
    * `end`: data dell'UPS dell'ultimo file di log da prelevare.

    Output:

    * Oggetto di tipo `StatisticsDto` che rappresenta il solo contenuto dei dati che verranno mandati come risposta alla richiesta *HTTP* gestita da `StatisticsController`.

4.1.4.10 StatisticsRepositoryPgSql

- **Descrizione**:
  - La classe implementa i metodi dell'interfaccia `StatisticsRepository` tramite una connessione al database effettuata dall'attributo `context`.
- **Attributi**:
  - `context`: oggetto di tipo `SmartLogContext` che rappresenta la connessione con il database.

4.1.4.11 StatisticsController

- **Metodi**:
  - `Statistics(startDateTime: DateTime, endDateTime: DateTime): IActionResult`

    Descrizione:
    - Gestisce una chiamata *HTTP GET* sull'endpoint `/api/statistics/{startDateTime}/{endDateTime}` e ritorna un oggetto di tipo `IActionResult`.

    Input:
    - `startDateTime`: data dell'UPS del primo file di log da prelevare;
    - `endDateTime`: data dell'UPS dell'ultimo file di log da prelevare.

    Output:
    - Oggetto di tipo `IActionResult` che rappresenta la risposta *HTTP*, la quale può avere esito positivo o negativo (vedi sezione API SmartLogStatistics).

4.1.4.12 StatisticsDto

- **Attributi**:
  - `Statistics`: oggetto di tipo `List<Statistic>` che rappresenta la lista di statistiche da restituire.

4.1.4.13 Statistic

- **Attributi**:
  - `Name`: variabile di tipo `string` che rappresenta il nome della statistica;
  - `Value`: variabile di tipo `double` che rappresenta il valore della statistica.

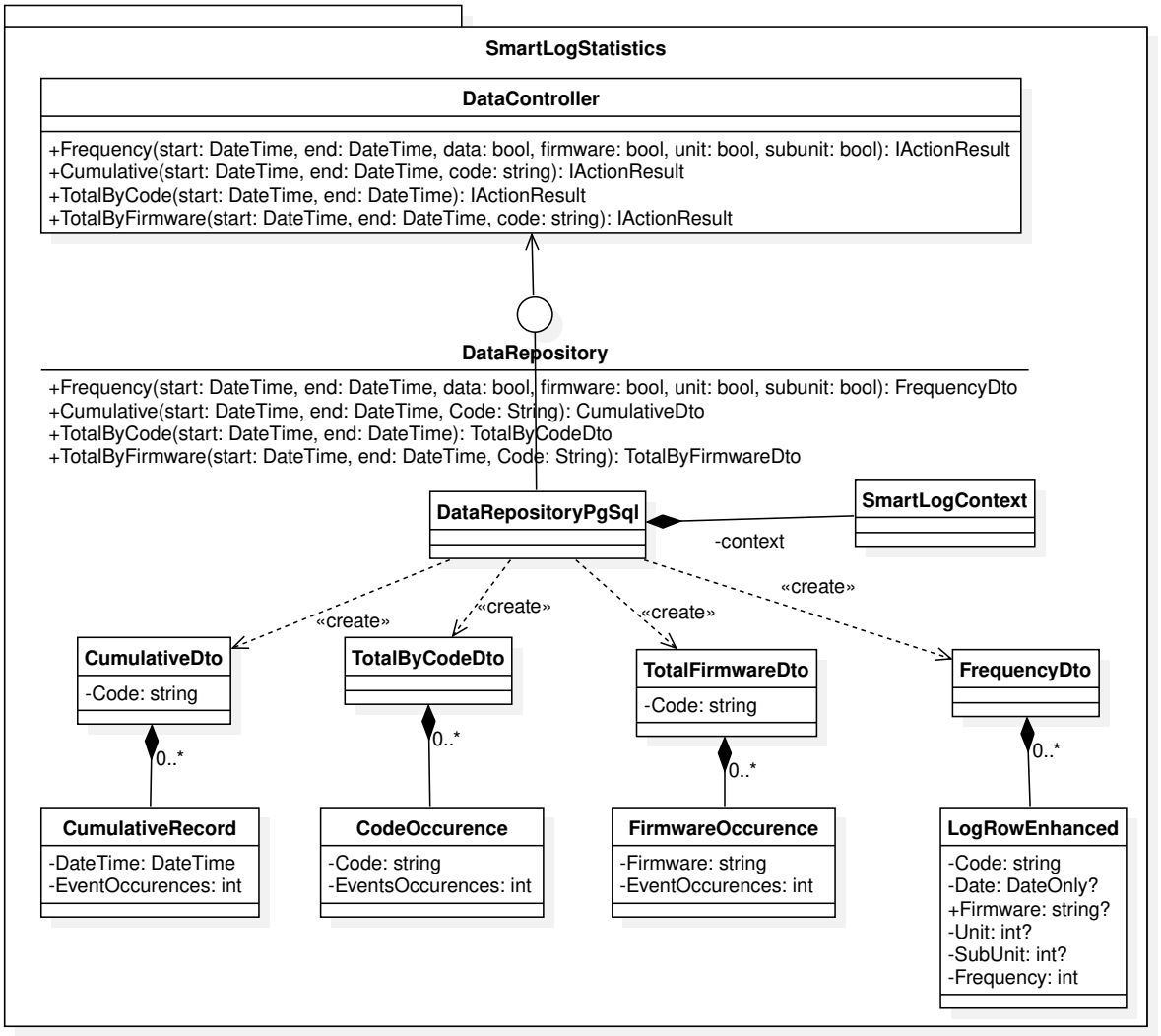

Figura 4.20: Diagramma delle classi per la gestione dei dati del progetto SmartLogStatistics.

4.1.4.14 DataRepository

- **Metodi**:
  - `Frequency(start: DateTime, end: DateTime, data: bool, firmware: bool, unit: bool, subunit: bool): FrequencyDto`

    Descrizione:
    - Collabora con il context per interagire con il database. Ritorna un oggetto di tipo `FrequencyDto`.

    Input:
    - `start`: data dell'UPS del primo file di log da prelevare;
    - `end`: data dell'UPS dell'ultimo file di log da prelevare;
    - `data`: variabile booleana che rappresenta il raggruppamento o meno per il campo data;

  - `firmware`: variabile booleana che rappresenta il raggruppamento o meno per il campo firmware;
  - `unit`: variabile booleana che rappresenta il raggruppamento o meno per il campo Unit;
  - `subUnit`: variabile booleana che rappresenta il raggruppamento o meno per il campo SubUnit.

  Output:

  - Oggetto di tipo `FrequencyDto` che rappresenta il solo contenuto dei dati che verranno mandati come risposta alla richiesta *HTTP* gestita da `DataController`.

- `Cumulative(start: DateTime, end: DateTime, code: string): CumulativeDto`

  Descrizione:

  - Collabora con il context per interagire con il database. Ritorna un oggetto di tipo `CumulativeDto`.

  Input:

  - `start`: data dell'UPS del primo file di log da prelevare;
  - `end`: data dell'UPS dell'ultimo file di log da prelevare;
  - `code`: il codice dell'evento di cui si vogliono ottenere le informazioni.

  Output:

  - Oggetto di tipo `CumulativeDto` che rappresenta il solo contenuto dei dati che verranno mandati come risposta alla richiesta *HTTP* gestita da `DataController`.

- `TotalByCode(start: DateTime, end: DateTime): TotalByCodeDto`

  Descrizione:

  - Collabora con il context per interagire con il database. Ritorna un oggetto di tipo `TotalByCodeDto`.

  Input:

  - `start`: data dell'UPS del primo file di log da prelevare;
  - `end`: data dell'UPS dell'ultimo file di log da prelevare.

  Output:

  - Oggetto di tipo `TotalByCodeDto` che rappresenta il solo contenuto dei dati che verranno mandati come risposta alla richiesta *HTTP* gestita da `DataController`.

- `TotalByFirmware(start: DateTime, end: DateTime, code: string): TotalByFirmwareDto`

  Descrizione:

  - Collabora con il context per interagire con il database. Ritorna un oggetto di tipo `TotalByFirmwareDto`.

Input:

* `start`: data dell'UPS del primo file di log da prelevare;
* `end`: data dell'UPS dell'ultimo file di log da prelevare;
* `code`: il codice dell'evento di cui si vogliono ottenere le informazioni.

Output:

* Oggetto di tipo `TotalByFirmwareDto` che rappresenta il solo contenuto dei dati che verranno mandati come risposta alla richiesta *HTTP* gestita da `DataController`.

– `Statistics(start: DateTime, end: DateTime): StatisticsDto`

Descrizione:

* Collabora con il context per interagire con il database. Ritorna un oggetto di tipo `StatisticsDto`.

Input:

* `start`: data dell'UPS del primo file di log da prelevare;
* `end`: data dell'UPS dell'ultimo file di log da prelevare.

Output:

* Oggetto di tipo `StatisticsDto` che rappresenta il solo contenuto dei dati che verranno mandati come risposta alla richiesta *HTTP* gestita da `StatisticsController`.

- **Attributi**:

  – `context`: oggetto di tipo `SmartLogContext` che rappresenta la connessione con il database.

4.1.4.15 DataRepositoryPgSql

- **Descrizione**:

  – La classe implementa i metodi dell'interfaccia `DataRepository` tramite una connessione al database effettuata dall'attributo `context`.

- **Attributi**:

  – `context`: oggetto di tipo `SmartLogContext` che rappresenta la connessione con il database.

4.1.4.16 DataController

- **Metodi**:

  – `Frequency(start: DateTime, end: DateTime, data: bool, firmware: bool, unit: bool, subunit: bool): IActionResult`

Descrizione:

  - Gestisce una chiamata *HTTP GET* sull'endpoint `/api/data/frequency/{start-DateTime}/{end-DateTime}?d={dataBool}&f={firmwareBool}&u={unitBool}&s={subunitBool}` e ritorna un oggetto di tipo `IActionResult`.

Input:

  - `start`: data dell'UPS del primo file di log da prelevare;
  - `end`: data dell'UPS dell'ultimo file di log da prelevare;
  - `data`: variabile booleana che rappresenta il raggruppamento o meno per il campo data;
  - `firmware`: variabile booleana che rappresenta il raggruppamento o meno per il campo firmware;
  - `unit`: variabile booleana che rappresenta il raggruppamento o meno per il campo Unit;
  - `subUnit`: variabile booleana che rappresenta il raggruppamento o meno per il campo SubUnit.

Output:

  - Oggetto di tipo `IActionResult` che rappresenta la risposta *HTTP*, la quale può avere esito positivo o negativo (vedi sezione API SmartLogStatistics).

- `Cumulative(start: DateTime, end: DateTime, code: string): IActionResult`

Descrizione:

  - Gestisce una chiamata *HTTP GET* sull'endpoint `/api/data/cumulative/{start-DateTime}/{end-DateTime}/{code}` e ritorna un oggetto di tipo `IActionResult`.

Input:

  - `start`: data dell'UPS del primo file di log da prelevare;
  - `end`: data dell'UPS dell'ultimo file di log da prelevare;
  - `code`: il codice dell'evento di cui si vogliono ottenere le informazioni.

Output:

  - Oggetto di tipo `IActionResult` che rappresenta la risposta *HTTP*, la quale può avere esito positivo o negativo (vedi sezione API SmartLogStatistics).

- `TotalByCode(start: DateTime, end: DateTime): IActionResult`

Descrizione:

  - Gestisce una chiamata *HTTP GET* sull'endpoint `/api/data/totalbycode/{start-DateTime}/{end-DateTime}` e ritorna un oggetto di tipo `IActionResult`.

Input:

  - `start`: data dell'UPS del primo file di log da prelevare;
  - `end`: data dell'UPS dell'ultimo file di log da prelevare.

Output:

    - Oggetto di tipo `IActionResult` che rappresenta la risposta *HTTP*, la quale può avere esito positivo o negativo (vedi sezione API SmartLogStatistics).

- `TotalByFirmware(start: DateTime, end: DateTime, code: string): IActionResult`

  Descrizione:

  - Gestisce una chiamata *HTTP GET* sull'endpoint `/api/data/totalbyfirmware/{start-DateTime}/{end-DateTime}/{code}` e ritorna un oggetto di tipo `IActionResult`.

  Input:

  - `start`: data dell'UPS del primo file di log da prelevare;
  - `end`: data dell'UPS dell'ultimo file di log da prelevare;
  - `code`: il codice dell'evento di cui si vogliono ottenere le informazioni.

  Output:

  - Oggetto di tipo `IActionResult` che rappresenta la risposta *HTTP*, la quale può avere esito positivo o negativo (vedi sezione).

4.1.4.17 TotalByCodeDto

- **Attributi**:
  - `_CodeOccurrence`: oggetto di tipo `List<CodeOccurrence>` che rappresenta la lista di codici degli eventi con la relativa frequenza di occorrenza.

4.1.4.18 CodeOccurence

- **Attributi**:
  - `Code`: variabile di tipo `string` che rappresenta il codice di cui si vuole conoscere la frequenza;
  - `NOccurences`: variabile di tipo `int` che rappresenta la frequenza.

4.1.4.19 TotalByFirmwareDto

- **Attributi**:
  - `Code`: variabile di tipo `string` che rappresenta il codice dell'evento;
  - `FirmwareOccurrences`: oggetto di tipo `List<FirmwareOccurrence>` che rappresenta la lista di codici dei firmware di un certo evento con la relativa frequenza di occorrenza.

4.1.4.20 FirmwareOccurence

- **Attributi**:
  - `Firmware`: variabile di tipo `string` che rappresenta il firmware di un certo evento di cui si vuole conoscere la frequenza;
  - `EventOccurences`: variabile di tipo `int` che rappresenta la frequenza.

4.1.4.21 FrequencyDto

- **Attributi**:
  - `LogRows`: oggetto di tipo `List<LogRowEnhanced>` che rappresenta la lista di eventi, con i campi secondo cui è stato eseguito il raggruppamento, e la loro frequenza di occorrenza.

4.1.4.22 LogRowEnhanced

- **Attributi**:
  - `Code`: variabile di tipo `string` che rappresenta il codice dell'evento;
  - `Date`: oggetto di tipo `DateTime` che rappresenta il momento in cui si è verificato l'evento;
  - `Firmware`: variabile di tipo `string` che rappresenta il firmware della macchina che ha scatenato l'evento;
  - `Unit`: variabile di tipo `int` che rappresenta la unit della macchina che ha scatenato l'evento;
  - `SubUnit`: variabile di tipo `int` che rappresenta la subUnit della macchina che ha scatenato l'evento;
  - `Frequency`: variabile di tipo `int` che rappresenta la frequenza di occorrenza dell'evento raggruppato secondo i campi desiderati.

4.1.4.23 CumulativeDto

- **Attributi**:
  - `Code`: variabile di tipo `string` che rappresenta il codice dell'evento;
  - `CumulativeRecords`: oggetto di tipo `List<CumulativeRecord>` che rappresenta la lista di momenti in cui si è verificato un certo evento.

4.1.4.24 CumulativeRecord

- **Attributi**:
  - `DateTime`: oggetto di tipo `DateTime` che rappresenta il momento in cui si è verificato un certo evento;
  - `EventOccurrences`: oggetto di tipo `int` che rappresenta la quantità di occorrenze di quell'evento fino al `DateTime`.

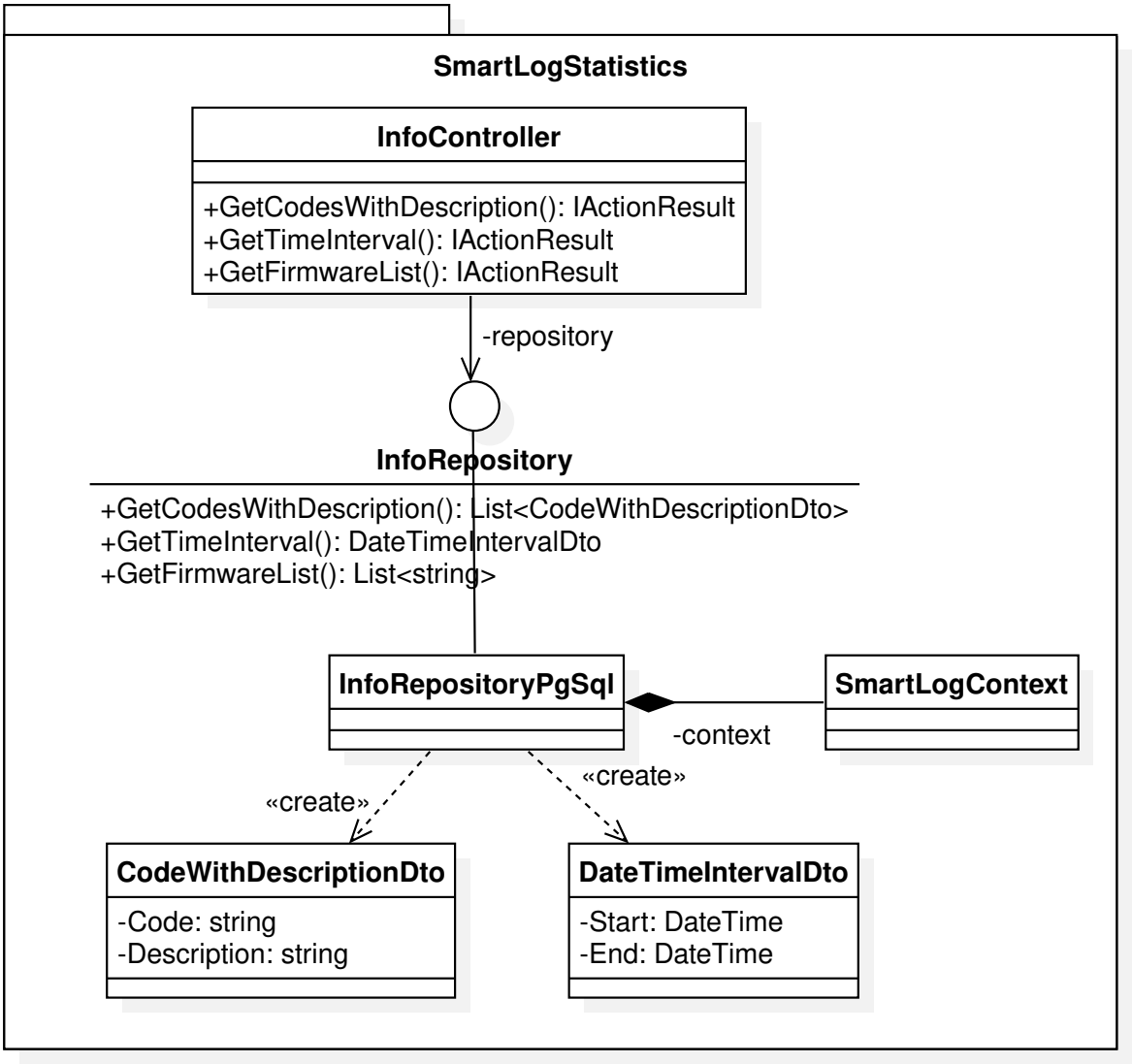

Figura 4.21: Diagramma delle classi per la gestione delle richieste di informazioni aggiuntive del progetto SmartLogStatistics.

4.1.4.25 InfoController

- **Metodi**:
  - `GetCodesWithDescription(): IActionResult`

    Descrizione:
    - Gestisce una chiamata *HTTP GET* sull'endpoint `/api/info/codes-description` e ritorna un oggetto di tipo `IActionResult`.

    Output:
    - Oggetto di tipo `IActionResult` che rappresenta la risposta *HTTP*, la quale può avere esito positivo o negativo (vedi sezione API SmartLogStatistics).
  - `GetTimeInterval(): IActionResult`

Descrizione:

* Gestisce una chiamata *HTTP GET* sull'endpoint `/api/info/timeinterval` e ritorna un oggetto di tipo `IActionResult`.

Output:

* Oggetto di tipo `IActionResult` che rappresenta la risposta *HTTP*, la quale può avere esito positivo o negativo (vedi sezione API SmartLogStatistics).

– `GetFirmwareList(): IActionResult`

Descrizione:

* Gestisce una chiamata *HTTP GET* sull'endpoint `/api/info/firmwarelist` e ritorna un oggetto di tipo `IActionResult`.

Output:

* Oggetto di tipo `IActionResult` che rappresenta la risposta *HTTP*, la quale può avere esito positivo o negativo (vedi sezione API SmartLogStatistics).

4.1.4.26 InfoRepository

- **Metodi**:

– `GetCodesWithDescription(): List<CodeWithDescriptionDto>`

Descrizione:

* Collabora con il context per interagire con il database.

Output:

* Oggetto di tipo `CodeWithDescriptionDto` che rappresenta il solo contenuto dei dati che verranno mandati come risposta alla richiesta *HTTP* gestita da `InfoController`.

– `GetTimeInterval(): DataTimeIntervalDto`

Descrizione:

* Collabora con il context per interagire con il database. Ritorna un oggetto di tipo `DateTimeIntervalDto`.

Output:

* Oggetto di tipo `DateTimeIntervalDto` che rappresenta il solo contenuto dei dati che verranno mandati come risposta alla richiesta *HTTP* gestita da `InfoController`.

– `GetFirmwareList(): List<string>`

Descrizione:

* Collabora con il context per interagire con il database. Ritorna un oggetto di tipo `List<string>`.

Output:

* Oggetto di tipo `List<string>` che rappresenta il solo contenuto dei dati che verranno mandati come risposta alla richiesta *HTTP* gestita da `InfoController`.

4.1.4.27 InfoRepositoryPgSql

- **Descrizione**:
  - La classe implementa i metodi dell'interfaccia `InfoRepository` tramite una connessione al database effettuata dall'attributo `context`.
- **Attributi**:
  - `context`: oggetto di tipo `SmartLogContext` che rappresenta la connessione con il database.

4.1.4.28 CodeWithDescriptionDto

- **Attributi**:
  - `Code`: variabile di tipo `string` che rappresenta il codice di cui si vuole conoscere la frequenza;
  - `Description`: variabile di tipo `string` che rappresenta la descrizione dell'evento.

4.1.4.29 CodeWithDescriptionDto

- **Attributi**:
  - `Start`: variabile di tipo `DateTime` che rappresenta il primo momento in cui si è verificato un evento;
  - `End`: variabile di tipo `DateTime` che rappresenta l'ultimo momento in cui si è verificato un evento.

4.1.5 Frontend SmartLogStatistics

4.1.5.1 Barra di navigazione

La barra di navigazione è l'elemento utilizzato per muoversi all'interno dell'applicazione; viene condivisa da tutte le schermate.

- **Percorsi:**
  - `/file-upload`;
  - `/statistics-table`;
  - `/event-table`;
  - `/cumulative-chart`;
  - `/histogram`;
  - `/pie-chart`.
- **Elementi:**
  - Nome dell'applicazione;

- Link alla schermata di caricamento file;
- Link alla schermata di visualizzazione delle statistiche generali;
- Link alla schermata di visualizzazione degli eventi raggruppati almeno per Code;
- Link alla schermata di visualizzazione del grafico cumulativo;
- Link alla schermata di visualizzazione dell'istogramma;
- Link alla schermata di visualizzazione del grafico a torta.

- **Architettura:**

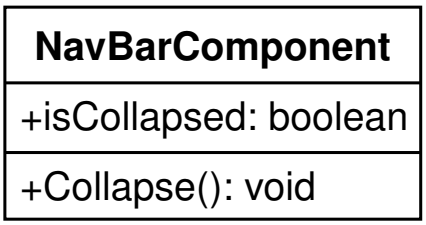

Figura 4.22: Diagramma delle classi del NavBarComponent.

- **NavBarComponent**
  Classe che gestisce il comportamento della navbar.

  Annotazioni:

  * `@Component`
    Viene definita la configurazione della classe.

  Attributi pubblici:

  + `isCollapsed: boolean`
    Indica quando la navbar è chiusa, è necessario per gestire il layout per schermi piccoli.

  Metodi pubblici:

  + `Collapse(): void`
    Metodo che si occupa di gestire l'apertura e la chiusura della navbar alla pressione del pulsante.

4.1.5.2 Schermata caricamento file

La schermata di caricamento file è la schermata che si vede all'avvio dell'applicazione; quando non è ancora stato caricato alcun file valido si vedono solamente il riquadro per il caricamento dei file e il pulsante di caricamento.
Durante il caricamento, viene mostrata una barra che indica la percentuale di completamento del caricamento dei file.
Una volta caricati i file, la schermata mostra anche l'esito del caricamento e i dettagli del caricamento per singolo file.

- **Percorso:**

  - `/file-upload`.

- **Elementi:**
  - Riquadro per il caricamento dei file, sia tramite drag and drop che tramite pulsante;
  - Pulsante per il caricamento dei file;
  - Barra con la percentuale di completamento del caricamento dei file;
  - Riquadro per la visualizzazione dell'esito del caricamento;
  - Sezione per la visualizzazione dei dettagli del caricamento per singolo file.
- **Requisiti soddisfatti:**
  - **RFS-1.1**: L'utente vuole caricare un insieme di file di log nel database;
  - **RFS-1.1.1**: L'utente visualizza un messaggio positivo in caso di successo nel caricamento di un insieme di file di log nel database;
  - **RFS-1.1.2**: L'utente ha caricato uno o più file già presenti nel database e quindi visualizza un messaggio di errore;
  - **RFS-1.1.3**: L'utente visualizza il progresso del caricamento dell'insieme dei file di log nel database;
  - **RFS-1.2**: L'utente ha selezionato uno o più file impossibili da leggere correttamente e quindi visualizza un messaggio di errore;
  - **RFS-1.2.1**: L'utente ha caricato uno o più file aventi un formato non corretto e quindi visualizza un messaggio di errore;
  - **RFS-1.2.2**: L'utente ha caricato uno o più file aventi alcuni dati che presentano un formato non corretto e quindi visualizza un messaggio di errore.
- **Architettura:**

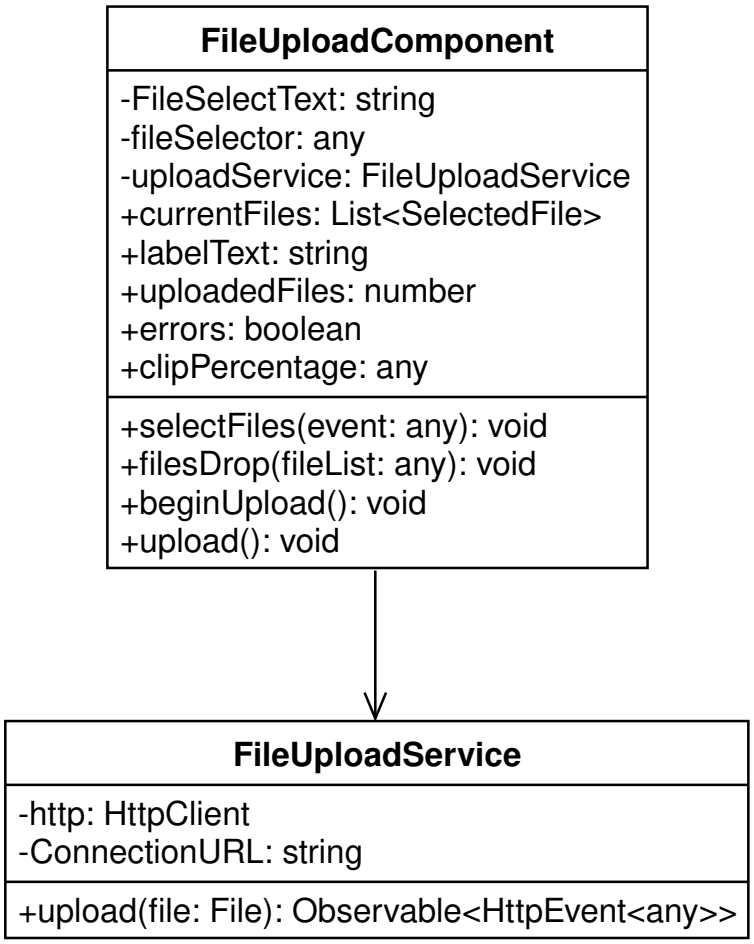

Figura 4.23: Diagramma delle classi del FileUploadComponent.

– **FileUploadComponent**
Classe che definisce il comportamento del widget di caricamento dei file.

Annotazioni:

* `@Component`
Viene definita la configurazione della classe.

Attributi privati:

– `FileSelectText: string`
Scritta di default per la selezione dei file;
– `@ViewChild("fileSelector") fileSelector: any;`
Gestore del controllo di input;
– `uploadService: FileUploadService`
Service che gestisce l'upload dei file sul server.

Attributi pubblici:

+ `currentFiles: List<SelectedFile>`
I file attualmente selezionati;
+ `labelText: string`
Testo della label sull'input;

+ `uploadedFiles: number`
  Numero di file attualmente caricati, usato anche come indice di avanzamento nella lista di file da caricare;
+ `errors: boolean`
  Indica alla vista se mostrare il messaggio di errore a caricamento completato;
+ `clipPercentage: any`
  Oggetto JSON che rispecchia lo stile della barra di avanzamento generale, serve per mascherare correttamente i colori del testo.

Metodi pubblici:

+ `selectFiles(event: any): void`
  Gestisce la selezione dei file tramite dialog;
+ `filesDrop(fileList: any): void`
  Gestisce il drag and drop del file;
+ `beginUpload(): void`
  Imposta lo stato corretto a tutti i file della lista e lancia il caricamento dei file sul db;
+ `upload(): void`
  Avvia il caricamento di un nuovo file sul db. In particolare utilizza l'indice this.uploadedFiles per scegliere il prossimo file da caricare.

– **FileUploadService**
Questo service si occupa di effettuare l'upload dei file di log.

Annotazioni:

* `@Injectable`
  Per poter essere identificato all'interno del sistema di dependency injection di Angular.

Attributi privati:

– `http: HttpClient`
  Il client http che si occupa di effettuare l'upload;
– `ConnectionURL: string`
  URL del backend.

Metodi pubblici:

+ `upload(file: File): Observable<HttpEvent<any>>`
  Effettua la chiamata al server.

4.1.5.3 Dialog di errore

Il dialog di errore è il dialog che viene visualizzato quando si verifica un errore. Il dialog contiene un titolo, il messaggio di errore e un pulsante per riprovare ad eseguire l'azione.

- **Percorsi:**
  – `/statistics-table`;
  – `/event-table`;

  - `/cumulative-chart`;
  - `/histogram`;
  - `/pie-chart`.

- **Elementi:**
  - Titolo dell'errore;
  - Messaggio di errore;
  - Pulsante per riprovare ad eseguire l'azione.

- **Requisiti soddisfatti:**
  - **RFS-1.4**: L'utente visualizza un messaggio di errore dopo aver inserito dei valori errati di data/ora per l'inizio o la fine dell'intervallo;
  - **RFS-1.4.1**: L'utente visualizza un messaggio di errore dopo aver inserito dei valori data/ora per l'inizio e la fine dell'intervallo logicamente errati per l'analisi degli eventi;
  - **RFS-1.4.2**: L'utente visualizza un messaggio di errore dopo aver inserito dei valori data/ora nel formato sbagliato per l'inizio o la fine dell'intervallo.

- **Architettura:**

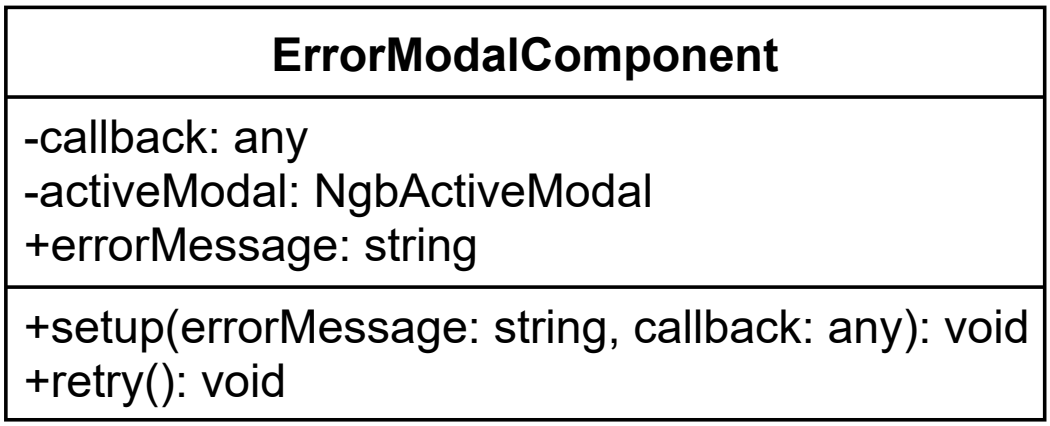

Figura 4.24: Diagramma delle classi del ErrorModalComponent.

  - **ErrorModalComponent**
    Classe per la gestione degli errori.

    Annotazioni:

    * `@Component`
      Viene definita la configurazione della classe.

    Attributi privati:

    - `callback: any`
      Funzione di callback;

– `activeModal: NgbActiveModal`
Modal attivo.

Attributi pubblici:

\+ `errorMessage: string`
Messaggio di errore.

Metodi pubblici:

\+ `setup(errorMessage: string, callback: any): void`
Setup del modal di errore;

\+ `retry(): void`
Metodo che chiude il modal aperto e chiama la funzione di callback.

4.1.5.4 Inserimento intervallo temporale

L'inserimento dell'intervallo temporale è necessario per visualizzare solo i dati relativi all'intervallo temporale specificato dall'utente; sarà quindi necessario inserire la data e l'ora di inizio e di fine dell'intervallo temporale e premere il pulsante di aggiornamento per visualizzare i dati relativi all'intervallo specificato.

- **Percorsi:**
  - `/statistics-table`;
  - `/histogram`.
- **Elementi:**
  - Casella per l'inserimento della data/ora iniziale;
  - Casella per l'inserimento della data/ora finale;
  - Pulsante per l'aggiornamento dei dati.
- **Requisiti soddisfatti:**
  - **RFS-1.3**: L'utente filtra gli eventi di cui vuole visualizzare le statistiche in base ad un range temporale.
- **Architettura:**

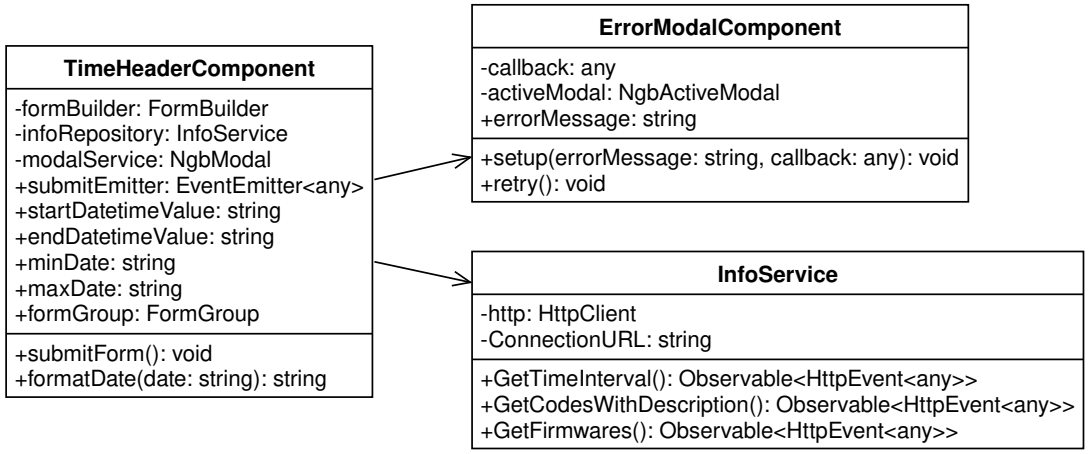

Figura 4.25: Diagramma delle classi del TimeHeaderComponent.

- **TimeHeaderComponent**
  Classe che crea un component in cui inserire un intervallo temporale.

  Annotazioni:

  * `@Component`
    Viene definita la configurazione della classe.

  Attributi privati:

  - `formBuilder: FormBuilder`
    Servizio di gestione dei form;
  - `infoRepository: InfoService`
    Servizio per ottenere le informazioni dal database per inizializzare il form;
  - `modalService: NgbModal`
    Servizio che si occupa di gestire i modal di bootstrap.

  Attributi pubblici:

  + `@Output() submitEmitter: EventEmitter<any>`
    Segnale per indicare che è avvenuto il submit;
  + `startDatetimeValue: string`
    Data del primo evento presente nel DB;
  + `endDatetimeValue: string`
    Data dell'ultimo evento presente nel DB;
  + `minDate: string`
    Data più piccola inseribile;
  + `maxDate: string`
    Data più grande inseribile;
  + `formGroup: FormGroup`
    Gestore del form.

  Metodi pubblici:

  + `submitForm(): void`
    Metodo che gestisce il submit del form;
  + `formatDate(date: string): string`
    Metodo per la formattazione di una data.

- **InfoService**
  Service per l'ottenimento dei dati di info dal backend.

  Annotazioni:

  * `@Injectable`
    Per poter essere identificato all'interno del sistema di dependency injection di Angular.

  Attributi privati:

  - `http: HttpClient`
    Il client http che effettua la chiamata al server;
  - `ConnectionURL: string`
    URL del backend.

Metodi pubblici:

+ `GetTimeInterval(): Observable<HttpEvent<any>>`
  Metodo che ottiene il minimo e il massimo DateTime nel database;
+ `GetCodesWithDescription(): Observable<HttpEvent<any>>`
  Metodo che ottiene la lista dei codici degli eventi con le relative descrizioni;
+ `GetFirmwares(): Observable<HttpEvent<any>>`
  Metodo che ottiene la lista dei firmware.

– ErrorModalComponent.

4.1.5.5 Inserimento intervallo temporale e campi per il ragguppamento

L'inserimento dell'intervallo temporale è necessario per visualizzare solo i dati relativi all'intervallo temporale specificato dall'utente; sarà quindi necessario inserire la data e l'ora di inizio e di fine dell'intervallo temporale. Per visualizzare gli eventi raggruppati bisognerà inserire anche i campi per i quali si vogliono raggruppare gli eventi (oltre al campo Code). Premendo il pulsante di aggiornamento si visualizzeranno gli eventi relativi all'intervallo specificato raggruppati per i campi selezionati.

- **Percorso:**
  - `/event-table`.
- **Elementi:**
  - Casella per l'inserimento della data/ora iniziale;
  - Casella per l'inserimento della data/ora finale;
  - Casella con scelta a tendina dei raggruppamenti da effettuare;
  - Pulsante per l'aggiornamento dei dati.
- **Requisiti soddisfatti:**
  - **RFS-1.3**: L'utente filtra gli eventi di cui vuole visualizzare le statistiche in base ad un range temporale;
  - **RFS-1.15**: L'utente vuole raggruppare i dati per uno o più campi;
  - **RFS-1.15.1**: L'utente vuole raggruppare i dati in base al valore del campo Code;
  - **RFS-1.15.2**: L'utente vuole raggruppare i dati in base al valore del campo data/ora;
  - **RFS-1.15.3**: L'utente vuole raggruppare i dati in base al valore della versione del firmware;
  - **RFS-1.15.4**: L'utente vuole raggruppare i dati in base al valore del campo Unit;
  - **RFS-1.15.5**: L'utente vuole raggruppare i dati in base al valore del campo SubUnit.

- **Architettura:**

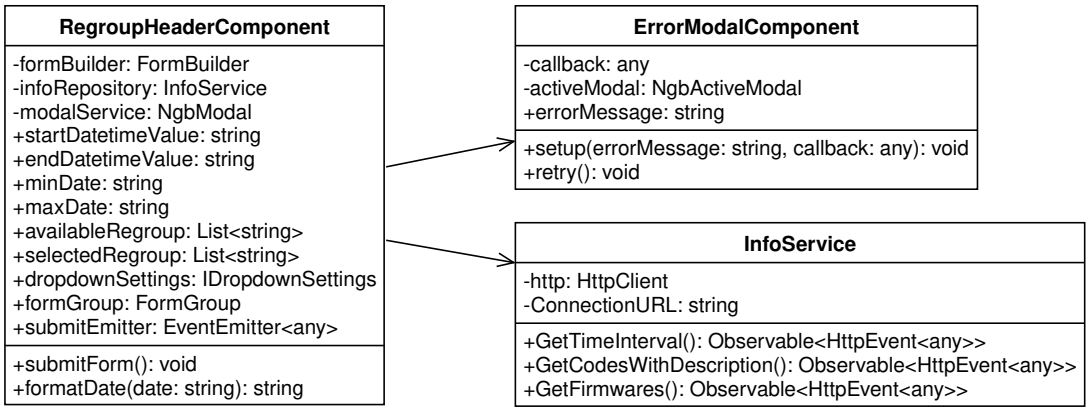

Figura 4.26: Diagramma delle classi del RegroupHeaderComponent.

– **RegroupHeaderComponent**
Classe che crea un component in cui inserire l'intervallo temporale e i campi di cui si vogliono ottenere i raggruppamenti.

Annotazioni:

* `@Component`
Viene definita la configurazione della classe.

Attributi privati:

– `formBuilder: FormBuilder`
Servizio di gestione dei form;
– `infoRepository: InfoService`
Servizio per ottenere le informazioni dal database;
– `modalService: NgbModal`
Servizio che si occupa di gestire i modal di bootstrap.

Attributi pubblici:

+ `startDatetimeValue: string`
Data del primo evento presente nel DB;
+ `endDatetimeValue: string`
Data dell'ultimo evento presente nel DB;
+ `minDate: string`
Data più piccola inseribile;
+ `maxDate: string`
Data più grande inseribile;
+ `availableRegroup: List<string>`
Lista dei raggruppamenti disponibili;
+ `selectedRegroup: List<string>`
Lista dei raggruppamenti selezionati;
+ `dropdownSettings: IDropdownSettings`
Impostazioni del menù a tendina;

+ `formGroup: FormGroup`
  Gestore del form;
+ `@Output() submitEmitter: EventEmitter<any>`
  Emitter dei dati del form.

Metodi pubblici:

+ `submitForm(): void`
  Metodo che gestisce il submit del form;
+ `formatDate(date: string): string`
  Metodo per la formattazione di una data.

– **InfoService**
Service per l'ottenimento dei dati di info dal backend.

Annotazioni:

* `@Injectable`
  Per poter essere identificato all'interno del sistema di dependency injection di Angular.

Attributi privati:

– `http: HttpClient`
  Il client http che effettua la chiamata al server;
– `ConnectionURL: string`
  URL del backend.

Metodi pubblici:

+ `GetTimeInterval(): Observable<HttpEvent<any>>`
  Metodo che ottiene il minimo e il massimo DateTime nel database;
+ `GetCodesWithDescription(): Observable<HttpEvent<any>>`
  Metodo che ottiene la lista dei codici degli eventi con le relative descrizioni;
+ `GetFirmwares(): Observable<HttpEvent<any>>`
  Metodo che ottiene la lista dei firmware.

– ErrorModalComponent.

4.1.5.6 Inserimento intervallo temporale e code

L'inserimento dell'intervallo temporale è necessario per visualizzare solo i dati relativi all'intervallo temporale specificato dall'utente; sarà quindi necessario inserire la data e l'ora di inizio e di fine dell'intervallo temporale. Si può inserire il code di interesse tra i code presenti nel database. Premendo il pulsante di aggiornamento si visualizzeranno i dati aggiornati.

- **Percorsi:**
  – `/cumulative-chart`;
  – `/pie-chart`.
- **Elementi:**
  – Casella per l'inserimento della data/ora iniziale;

- Casella per l'inserimento della data/ora finale;
- Casella con scelta a tendina del code;
- Pulsante per l'aggiornamento dei dati.

- **Requisiti soddisfatti:**
  - **RFS-1.3**: L'utente filtra gli eventi di cui vuole visualizzare le statistiche in base ad un range temporale.

- **Architettura:**

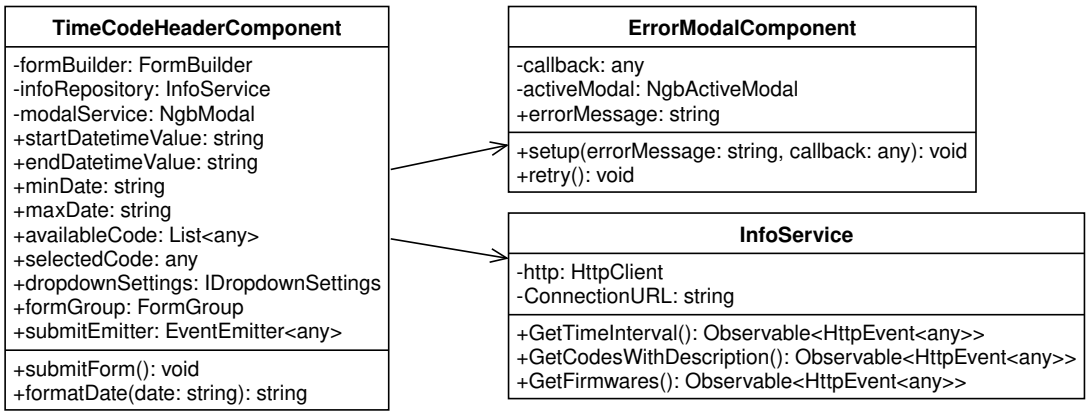

Figura 4.27: Diagramma delle classi del TimeCodeHeaderComponent.

- **TimeCodeHeaderComponent**
  Classe che crea un component in cui inserire l'intervallo temporale e il Code di cui si vogliono ottenere i dati.

  Annotazioni:

  * `@Component`
    Viene definita la configurazione della classe.

  Attributi privati:

  - `formBuilder: FormBuilder`
    Servizio di gestione dei form;
  - `infoRepository: InfoService`
    Servizio per ottenere le informazioni dal database;
  - `modalService: NgbModal`
    Servizio che si occupa di gestire i modal di bootstrap.

  Attributi pubblici:

  + `startDatetimeValue: string`
    Data del primo evento presente nel DB;
  + `endDatetimeValue: string`
    Data dell'ultimo evento presente nel DB;
  + `minDate: string`
    Data più piccola inseribile;

+ `maxDate: string`
  Data più grande inseribile;
+ `availableCode: List<any>`
  Lista dei Code disponibili;
+ `selectedCode: any`
  Code selezionato;
+ `dropdownSettings: IDropdownSettings`
  Impostazioni del menù a tendina;
+ `formGroup: FormGroup`
  Gestore del form;
+ `@Output() submitEmitter: EventEmitter<any>`
  Segnale per indicare che è avvenuto il submit.

Metodi pubblici:

+ `submitForm(): void`
  Metodo che gestisce il submit del form;
+ `formatDate(date: string): string`
  Metodo per la formattazione di una data.

– **InfoService**
Service per l'ottenimento dei dati di info dal backend.

Annotazioni:

* `@Injectable`
  Per poter essere identificato all'interno del sistema di dependency injection di Angular.

Attributi privati:

– `http: HttpClient`
  Il client http che effettua la chiamata al server;
– `ConnectionURL: string`
  URL del backend.

Metodi pubblici:

+ `GetTimeInterval(): Observable<HttpEvent<any>>`
  Metodo che ottiene il minimo e il massimo DateTime nel database;
+ `GetCodesWithDescription(): Observable<HttpEvent<any>>`
  Metodo che ottiene la lista dei codici degli eventi con le relative descrizioni;
+ `GetFirmwares(): Observable<HttpEvent<any>>`
  Metodo che ottiene la lista dei firmware.

– ErrorModalComponent.

4.1.5.7 Tabella delle statistiche generali

La pagina della tabella delle statistiche generali contiene un form in cui inserire l'intervallo temporale e una tabella in cui vengono visualizzate le statistiche generali. Le statistiche visualizzate nella tabella sono:

- Numero di file;

- Massimo numero di eventi per file;
- Numero medio di eventi per file;
- Deviazione standard del numero di eventi per file.
- **Percorso:**
  - `/statistics-table`.
- **Elementi:**
  - Form per l'inserimento dell'intervallo temporale;
  - Tabella per la visualizzazione delle statistiche generali.
- **Requisiti soddisfatti:**
  - **RFS-1.6**: L'utente visualizza il numero di storici analizzati, la media di eventi per file di log, il massimo numero di eventi per file di log e la deviazione standard sul numero di eventi per file di log in forma tabellare;
  - **RFS-1.6.1**: L'utente visualizza il numero di storici analizzati in forma tabellare;
  - **RFS-1.6.2**: L'utente visualizza la media di eventi per file di log in forma tabellare;
  - **RFS-1.6.3**: L'utente visualizza il massimo numero di eventi per file di log in forma tabellare;
  - **RFS-1.6.4**: L'utente visualizza la deviazione standard sul numero di eventi per file di log in forma tabellare.
- **Architettura:**

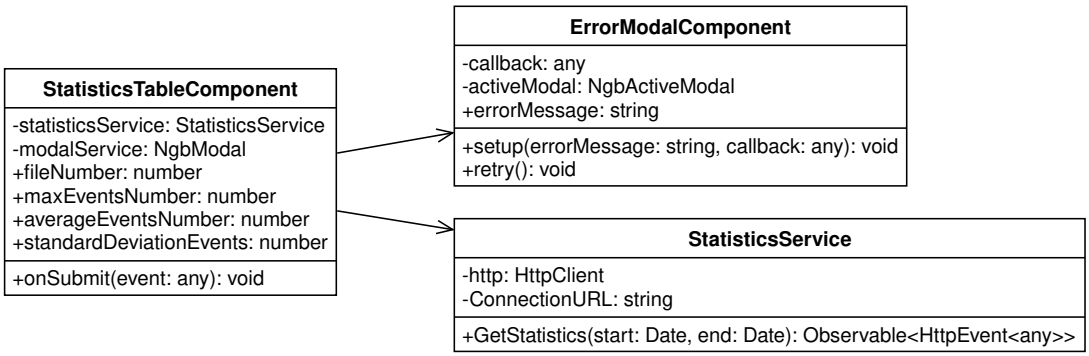

Figura 4.28: Diagramma delle classi del StatisticsTableComponent.

- **StatisticsTableComponent**
  Classe che definisce il comportamento della tabella delle statistiche.

  Annotazioni:

  * `@Component`
    Viene definita la configurazione della classe.

Attributi privati:

- `statisticsService: StatisticsService`
  Servizio di fetch delle statistiche;
- `modalService: NgbModal`
  Dialog di errore.

Attributi pubblici:

+ `fileNumber: number`
  Numero di file del periodo in analisi;
+ `maxEventsNumber: number`
  Massimo numero di eventi per file del periodo in analisi;
+ `averageEventsNumber: number`
  Numero medio di eventi per file del periodo in analisi;
+ `standardDeviationEvents: number`
  Deviazione standard del numero di eventi per file del periodo in analisi.

Metodi pubblici:

+ `onSubmit(event: any): void`
  Gestisce la richiesta al back-end per ottenere le statistiche.

– **StatisticsService**
Service per l'ottenimento delle statistiche dal backend.

Annotazioni:

* `@Injectable`
  Per poter essere identificato all'interno del sistema di dependency injection di Angular.

Attributi privati:

- `http: HttpClient`
  Il client http che effettua la chiamata al server;
- `ConnectionURL: string`
  URL del backend.

Metodi pubblici:

+ `GetStatistics(start: Date, end: Date): Observable<HttpEvent<any>>`
  Metodo che ottiene le statistiche comprese tra il lower bound e l'upper bound dell'intervallo di ricerca.

– ErrorModalComponent.

4.1.5.8 Tabella degli eventi raggruppati

La pagina della tabella degli eventi raggruppati contiene un form in cui inserire l'intervallo temporale e i campi per cui si vogliono raggruppare gli eventi e una tabella in cui vengono visualizzati gli eventi raggruppati per i campi selezionati.

- **Percorso:**
  - `/event-table`.

- **Elementi:**
  - Form per l'inserimento dell'intervallo temporale e dei campi per il ragguppamento;
  - Tabella per la visualizzazione degli eventi raggruppati.
- **Requisiti soddisfatti:**
  - **RFS-1.7**: L'utente visualizza in una tabella una lista di eventi raggruppati per Code;
  - **RFS-1.7.1**: L'utente visualizza in una tabella una lista di eventi raggruppati per data;
  - **RFS-1.7.2**: L'utente visualizza in una tabella una lista di eventi raggruppati per versione del firmware;
  - **RFS-1.7.3**: L'utente visualizza in una tabella una lista di eventi raggruppati per Unit;
  - **RFS-1.7.4**: L'utente visualizza in una tabella una lista di eventi raggruppati per SubUnit;
  - **RFS-1.8**: L'utente vuole ordinare gli eventi secondo il valore di un campo dati;
  - **RFS-1.8.1**: L'utente vuole ordinare gli eventi per frequenza di occorrenza;
  - **RFS-1.8.2**: L'utente vuole ordinare gli eventi in base al valore del campo Unit;
  - **RFS-1.8.3**: L'utente vuole ordinare gli eventi in base al valore del campo SubUnit;
  - **RFS-1.8.4**: L'utente vuole ordinare gli eventi in base al valore della versione firmware;
  - **RFS-1.8.5**: L'utente vuole ordinare gli eventi in base al valore della data;
  - **RFS-1.9**: L'utente vuole filtrare gli eventi in base al valore di un campo dati;
  - **RFS-1.9.1**: L'utente vuole filtrare gli eventi in base al valore del campo Unit;
  - **RFS-1.9.2**: L'utente vuole filtrare gli eventi in base al valore del campo SubUnit;
  - **RFS-1.9.3**: L'utente vuole filtrare gli eventi in base al valore della versione firmware;
  - **RFS-1.16**: L'utente visualizza un avviso dovuto all'assenza di dati da visualizzare.
- **Architettura:**

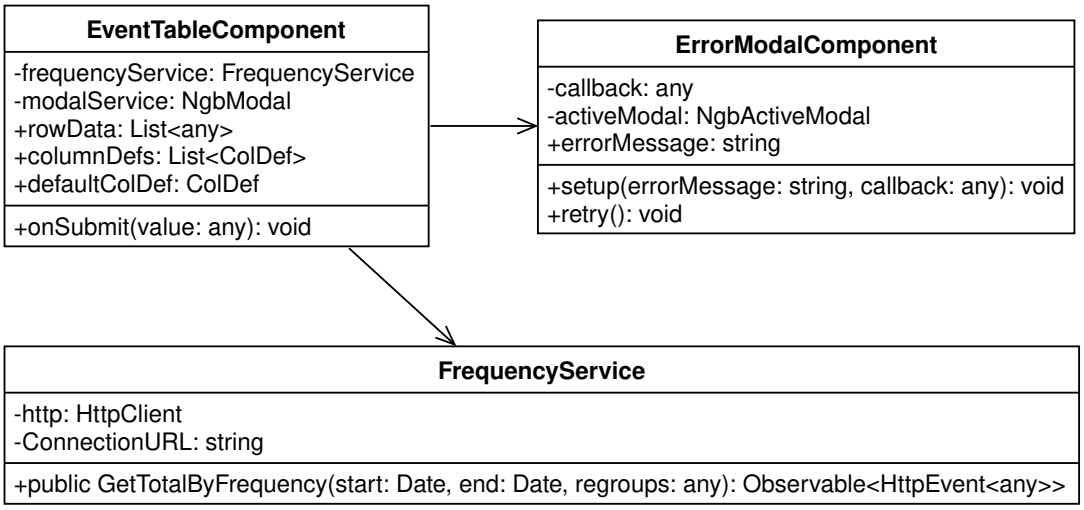

Figura 4.29: Diagramma delle classi del EventTableComponent.

– **EventTableComponent**
Classe che definisce il comportamento della tabella di visualizzazione degli eventi raggruppati almeno per code.

Annotazioni:

* `@Component`
Viene definita la configurazione della classe.

Attributi privati:

– `frequencyService : FrequencyService`
Servizio che ottiene i dati dal backend;
– `modalService: NgbModal`
Dialog di errore.

Attributi pubblici:

+ `rowData: List<any>`
Elenco dei dati da mostrare nella tabella;
+ `columnDefs: List<ColDef>`
Definizione dei campi dati da mostrare nella tabella;
+ `defaultColDef: ColDef`
Impostazioni di default dei campi della tabella.

Metodi pubblici:

+ `onSubmit(value: any): void`
Metodo che gestisce il submit del form.

– **FrequencyService**
Servizio per l'ottenimento dal backend di un JSON che rappresenta gli eventi nell'intervallo temporale dato, raggruppati per i campi specificati.

Annotazioni:

* `@Injectable`
  Per poter essere identificato all'interno del sistema di dependency injection di Angular.

Attributi privati:

- `http: HttpClient`
  Il client http che effettua la chiamata al server;
- `ConnectionURL: string`
  URL del backend.

Metodi pubblici:

+ `GetTotalByFrequency(start: Date, end: Date, regroups: any): Observable<HttpEvent<any>>`
  Metodo che ottiene tramite richesta http un JSON che rappresenta gli eventi nell'intervallo temporale dato, raggruppati per i campi specificati.

- ErrorModalComponent.

4.1.5.9 Grafico cumulativo

La pagina del grafico cumulativo contiene un form in cui inserire l'intervallo temporale e il code di cui si vuole ottenere i dati e un grafico cumulativo del code selezionato.

- **Percorso:**
  - `/cumulative-chart`.
- **Elementi:**
  - Form per l'inserimento dell'intervallo temporale e del code;
  - Grafico cumulativo del code selezionato.
- **Requisiti soddisfatti:**
  - **RFS-1.11**: L'utente visualizza il numero totale di occorrenze di un singolo evento in un intervallo tempo in un grafico cumulativo in cui nell'asse x è riportato il tempo e nell'asse y è riportata la somma delle occorrenze dell'evento;
  - **RFS-1.16**: L'utente visualizza un avviso dovuto all'assenza di dati da visualizzare.
- **Architettura:**

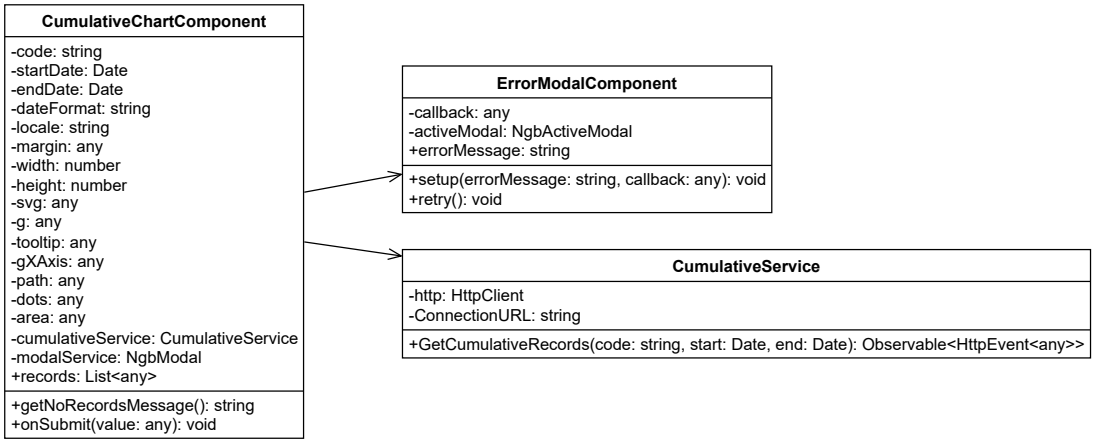

Figura 4.30: Diagramma delle classi del CumulativeChartComponent.

– **CumulativeChartComponent**
Classe per la rappresentazione dei dati in un grafico cumulativo.

Annotazioni:

* `@Component`
Viene definita la configurazione della classe.

Attributi privati:

– `code: string`
Codice dell'evento;
– `startDate: Date`
Data di lower-bound del intervallo temporale preso in esame;
– `endDate: Date`
Data di upper-bound del intervallo temporale preso in esame;
– `dateFormat: string`
Formato della data usato per la formattazione;
– `locale: string`
Locale usato per formattare le date;
– `margin: any`
Margine del grafico;
– `width: number`
Larghezza del grafico;
– `height: number`
Altezza del grafico;
– `svg: any`
Il grafico da disegnare a schermo;
– `g: any`
Il grafico da disegnare;
– `tooltip: any`
Il tooltip mostrato quando si passa sopra un punto;
– `gXAxis: any`
L'asse x;

- `path: any`
  La linea disegnata sul grafico;
- `dots: any`
  I punti disegnati sul grafico;
- `area: any`
  L'area colorata sotto la linea del grafico;
- `cumulativeService: CumulativeService`
  Servizio che ottiene i dati dal backend;
- `modalService: NgbModal`
  Dialog di errore.

Attributi pubblici:

+ `records: List<any>`
  I dati da visualizzare nel grafico.

Metodi pubblici:

+ `getNoRecordsMessage(): string`
  Metodo per l'ottenimento del messaggio da visualizzare in caso di mancanza di dati;
+ `onSubmit(value: any): void`
  Metodo che gestisce il submit del form.

– **CumulativeService**
Servizio per l'ottenimento dal backend di un JSON che rappresenta il numero di occorrenze degli eventi nell'intervallo temporale dato in modo cumulativo.

Annotazioni:

* `@Injectable`
  Per poter essere identificato all'interno del sistema di dependency injection di Angular.

Attributi privati:

- `http: HttpClient`
  Il client http che effettua la chiamata al server;
- `ConnectionURL: string`
  URL del backend.

Metodi pubblici:

+ `GetCumulativeRecords(code: string, start: Date, end: Date): Observable<HttpEvent<any>>`
  Invoca una richiesta HTTP per prelevare l'andamento cumulativo di un evento dato un intervallo temporale.

– ErrorModalComponent.

4.1.5.10 Istogramma

La pagina dell'istogramma contiene un form in cui inserire l'intervallo temporale e un istogramma in cui si visualizza il numero di occorrenze per ogni code.

- **Percorso:**

- `/histogram`.

- **Elementi:**
  - Form per l'inserimento dell'intervallo temporale;
  - Istogramma.

- **Requisiti soddisfatti:**
  - **RFS-3.12**: L'utente visualizza il numero totale di occorrenze degli eventi in un intervallo tempo in un istogramma in cui nell'asse x sono riportate le occorrenze di ogni evento nell'intervallo di tempo e nell'asse y sono riportati gli eventi;
  - **RFS-1.16**: L'utente visualizza un avviso dovuto all'assenza di dati da visualizzare.

- **Architettura:**

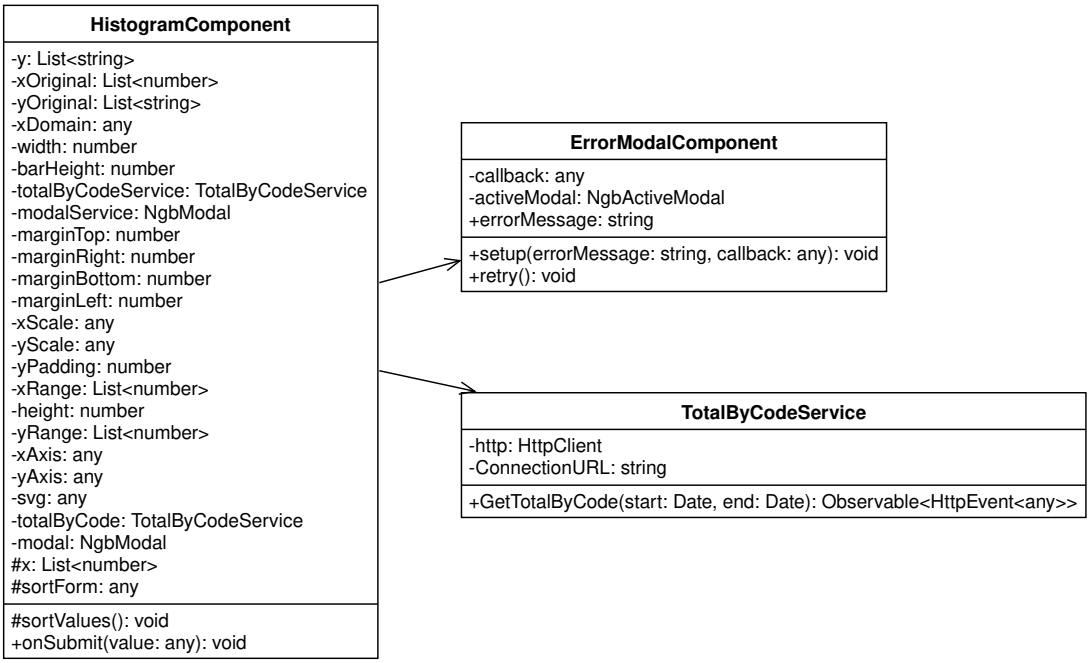

Figura 4.31: Diagramma delle classi del HistogramComponent.

- **HistogramComponent**
  Classe per la rappresentazione dei dati in un istogramma.

  Annotazioni:

  * `@Component`
    Viene definita la configurazione della classe.

  Attributi privati:

  - `y: List<string>`
    Variabile contenente i nomi dei vari codici;

- `xOriginal: List<number>`
  Variabile che contiene l'ordine originale dei dati della frequenza per utilizzarla quando non si usa nessun ordinamento;
- `yOriginal: List<string>`
  Variabile che contiene l'ordine originale dei codici per utilizzarla quando non si usa nessun ordinamento;
- `xDomain: any`
  Minimo (0) e massimo delle frequenza;
- `width: number`
  Larghezza totale dell'svg;
- `barHeight: number`
  Spessore di ogni singola barra;
- `totalByCodeService: TotalByCodeService`
  Il service che fornisce i dati;
- `modalService: NgbModal`
  Il dialog che serve per segnalare errori;
- `marginTop: number`
  Margine superiore dell'svg;
- `marginRight: number`
  Margine destro dell'svg;
- `marginBottom: number`
  Margine inferiore dell'svg;
- `marginLeft: number`
  Margine sinistro dell'svg;
- `xScale: any`
  Scala per posizionare i valori rispetto all'asse orizzontale;
- `yScale: any`
  Scala per posizionare i codici sull'asse verticale;
- `yPadding: number`
  Distanza fra una barra e l'altra;
- `xRange: List<number>`
  Range di valori presenti sull'asse x;
- `height: number`
  Altezza totale dell'svg;
- `yRange: List<number>`
  Range di valori dell'asse y;
- `xAxis: any`
  L'effettivo asse x;
- `yAxis: any`
  L'effettivo asse y;
- `svg: any`
  L'effettivo svg;
- `totalByCode: TotalByCodeService`
  Il service per ottenere i dati dall'API;
- `modal: NgbModal`
  Il pop-up per segnalare un errore nel fetch dei dati dall'API.

Attributi protetti:

`x: List<number>`
Variabile contenente i valori della frequenza dei vari codici;

`sortForm: any`
Il form group contenente il form di ordinamento dei dati.

Metodi protetti:

`sortValues(): void`
Metodo che ordina i dati.

Metodi pubblici:

+ `onSubmit(value:any): void`
Metodo che gestisce il submit del form.

– **TotalByCodeService**
Servizio per l'ottenimento dal backend di un JSON contenente i codici con le relative frequenze.

Annotazioni:

∗ `@Injectable`
Per poter essere identificato all'interno del sistema di dependency injection di Angular.

Attributi privati:

– `http: HttpClient`
Il client http che effettua la chiamata al server;

– `ConnectionURL: string`
URL del backend.

Metodi pubblici:

+ `GetTotalByCode(start: Date, end: Date): Observable<HttpEvent<any>>`
Metodo che ottiene una lista di codici con le relative frequenze.

– ErrorModalComponent.

4.1.5.11 Grafico a torta

La pagina del grafico a torta contiene un form in cui inserire l'intervallo temporale e il code di cui si vuole ottenere i dati e un grafico a torta in cui si visualizzano le percentuali di occorrenze del code selezionato per ogni firmware avente almeno un'occorrenza di evento con il Code scelto.

- **Percorso:**
  - `/pie-chart`.
- **Elementi:**
  - Form per l'inserimento dell'intervallo temporale e del code;
  - Grafico a torta dei dati del code selezionato.

- **Requisiti soddisfatti:**
  - **RFS-1.13**: L'utente visualizza un grafico a torta con il numero di occorrenze di un evento raggruppate per versioni firmware;
  - **RFS-1.16**: L'utente visualizza un avviso dovuto all'assenza di dati da visualizzare.
- **Architettura:**

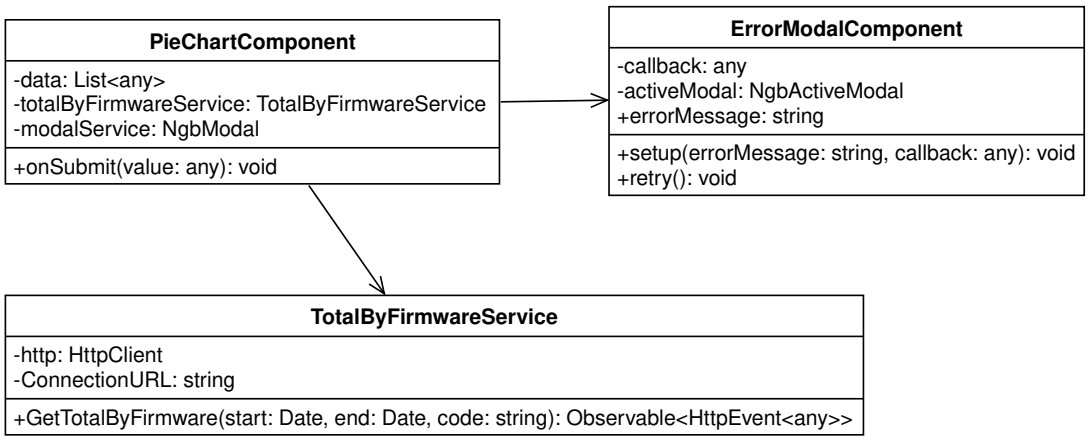

Figura 4.32: Diagramma delle classi del PieChartComponent.

- **PieChartComponent**
  Classe per la rappresentazione dei dati in un grafico a torta.

  Annotazioni:

  * `@Component`
    Viene definita la configurazione della classe.

  Attributi privati:

  - `totalByFirmwareService: TotalByFirmwareService`
    Servizio che ottiene i dati dal backend;
  - `modalService: NgbModal`
    Dialog di errore.

  Attributi protetti:

  # `data: List<any>`
    Dati da rappresentare nel grafico.

  Metodi pubblici:

  + `onSubmit(value: any): void`
    Metodo che gestisce il submit del form.
- **TotalByFirmwareService**
  Servizio per l'ottenimento dal backend di un JSON che rappresenta il numero di occorrenze dell'evento selezionato, comprese nell'intervallo temporale dato, raggruppate per versione firmware.

  Annotazioni:

* `@Injectable`
  Per poter essere identificato all'interno del sistema di dependency injection di Angular.

Attributi privati:

- `http: HttpClient`
  Il client http che effettua la chiamata al server;
- `ConnectionURL: string`
  URL del backend.

Metodi pubblici:

+ `GetTotalByFirmware(start: Date, end: Date, code: string): Observable<HttpEvent<any>>`
  Metodo che ottiene tramite richesta http un JSON che rappresenta il numero di occorrenze dell'evento selezionato, comprese nell'intervallo temporale dato, raggruppate per versione firmware.

– ErrorModalComponent.

4.2 Database

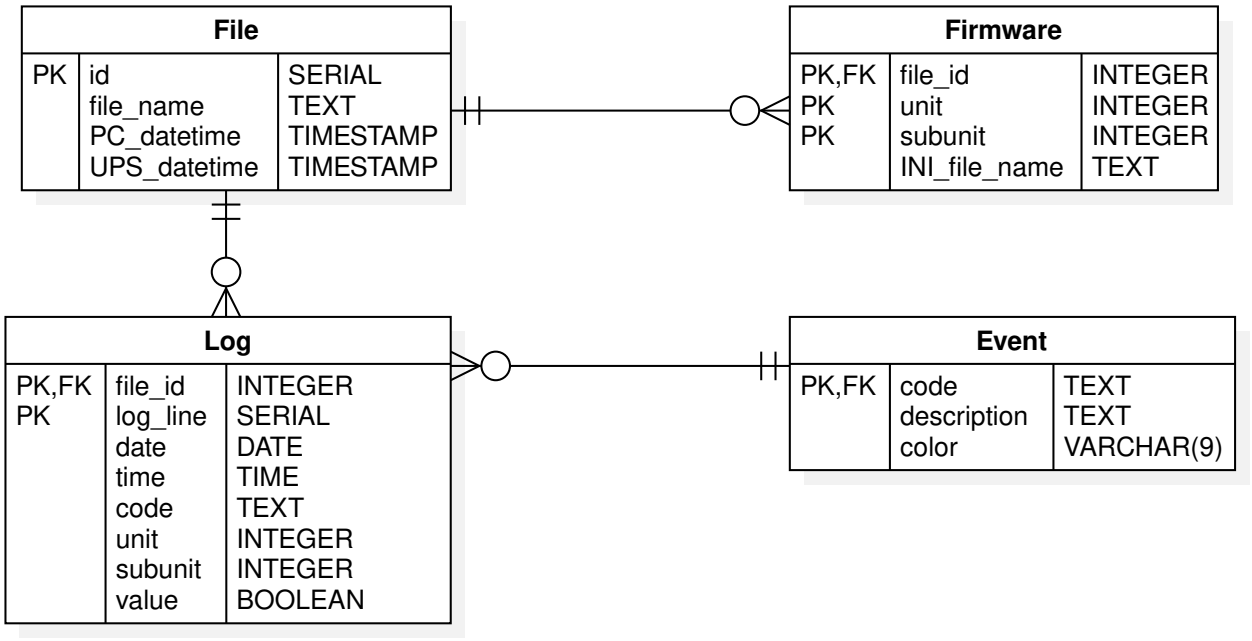

Figura 4.33: Architettura del Database.

4.2.1 Entità

Il database è composto da 4 entità:

- `File`: rappresenta un singolo file di log. Composto da:
  - `id`: identificativo numerico univoco di un file di log;

  - `file_name`: nome del file di log;
  - `PC_datetime`: momento in cui è stato ricevuto il file di log dal computer;
  - `UPS_datetime`: momento in cui è stato prodotto il file di log dall'UPS.

- `Firmware`: rappresenta le versioni del firmware di ogni subunit in un certo file di log. Composto da:
  - `file_id`: identificativo del file a cui appartiene;
  - `unit`: numero della Unit che contiene la SubUnit indicata;
  - `subunit`: numero della SubUnit che utilizza questa versione del firmware;
  - `INI_file_name`: nome della versione del firmware.

- `Log`: rappresenta gli effettivi dati di log di un file. Composto da:
  - `file_id`: identificativo del file a cui appartiene;
  - `log_line`: numero della linea del log;
  - `date`: data in cui si è scatenato l'evento;
  - `time`: ora in cui si è scatenato l'evento;
  - `code`: codice dell'evento scatenato;
  - `unit`: unità in cui si è scatenato l'evento;
  - `subunit`: sotto-unità in cui si è scatenato l'evento;
  - `value`: valore dell'evento scatenato.

- `Event`: rappresenta i vari eventi che sono stati scatenati con i relativi dettagli. Composto da:
  - `code`: codice dell'evento scatenato;
  - `description`: descrizione dell'evento scatenato;
  - `color`: colore che rappresenta l'evento scatenato.

4.2.2 Query

Le operazioni effettuate sulla base di dati sono le seguenti:

- Caricamento di uno o più file di log, con conseguente aggiornamento dei valori presenti in tutte le tabelle (a cascata: `File`, `Log`, `Firmware` ed eventualmente `Event`);

- Ottenimento di statistiche sui dati presenti nel database, quali:
  - Numero di storici analizzati;
  - Media di eventi per file di log;
  - Massimo numero di eventi per file di log;
  - Deviazione standard sul numero di eventi per file di log.

- Ottenimento della lista dei codici degli eventi e delle loro descrizioni (dalla tabella `Event`);

- Ottenimento di informazioni su un sottoinsieme di occorrenze di eventi presenti nel database raggruppati almeno per codice dell'evento (quali, ad esempio, frequenza di occorrenza e numero di occorrenze per evento).

4.3 Struttura

L'architettura utilizzata da entrambi gli applicativi è la `Layered Architecture`. L'utilizzo di una Layered Architecture consente di organizzare l'applicazione in moduli distinti, facilitando la manutenzione, il testing e lo sviluppo parallelo. Ogni strato ha un compito ben definito e comunica solo con gli strati adiacenti, creando una separazione delle responsabilità e riducendo la complessità complessiva del sistema.

4.3.1 Persistance Layer

Il `Persistance Layer` si occupa di tutte le operazioni di accesso dei dati in memoria e nel database. Nell'applicativo *SmartLogStatistics* è presente un database che permette di gestire la persistenza dei dati estrapolati dai file di log. Nell'applicativo *SmartLogViewer*, invece, questo layer si occupa di caricare il file di log in memoria.

4.3.2 Business Layer

Il `Business Layer` è responsabile della logica di business dell'applicazione. Nell'applicativo *SmartLogStatistics* questo strato comprende il backend che manipola i dati e calcola le statistiche richieste. Mentre nell'applicativo *SmartLogViewer* questo layer è implementato nel front-end. Qui avviene l'elaborazione dei dati provenienti dal `Persistance Layer` e la generazione delle statistiche desiderate.

4.3.3 Application Layer

L'`Application Layer` svolge un ruolo di intermediazione tra il `Business Layer` e il `Presentation Layer`. In entrambi gli applicativi, l'`Application Layer` è implementato attraverso i file TypeScript del frontend. Questo strato gestisce le richieste dell'utente provenienti dal `Presentation Layer`, invoca le opportune funzionalità del `Business Layer` e restituisce i risultati al `Presentation Layer` per la visualizzazione.

4.3.4 Presentation Layer

Il `Presentation Layer` è il front-end dell'applicazione. È responsabile dell'interfaccia utente e della presentazione dei dati all'utente. In entrambi gli applicativi, il `Presentation Layer` comprende i componenti che gestiscono la visualizzazione dei dati e l'interazione dell'utente, come le pagine web o l'interfaccia grafica.

4.4 Design Pattern Utilizzati

Di seguito sono elencati i design patter utilizzati nella realizzazione del progetto, in ordine alfabetico. Le immagini dove presenti sono prese dal sito `https://refactoring.guru`.

4.4.1 Command

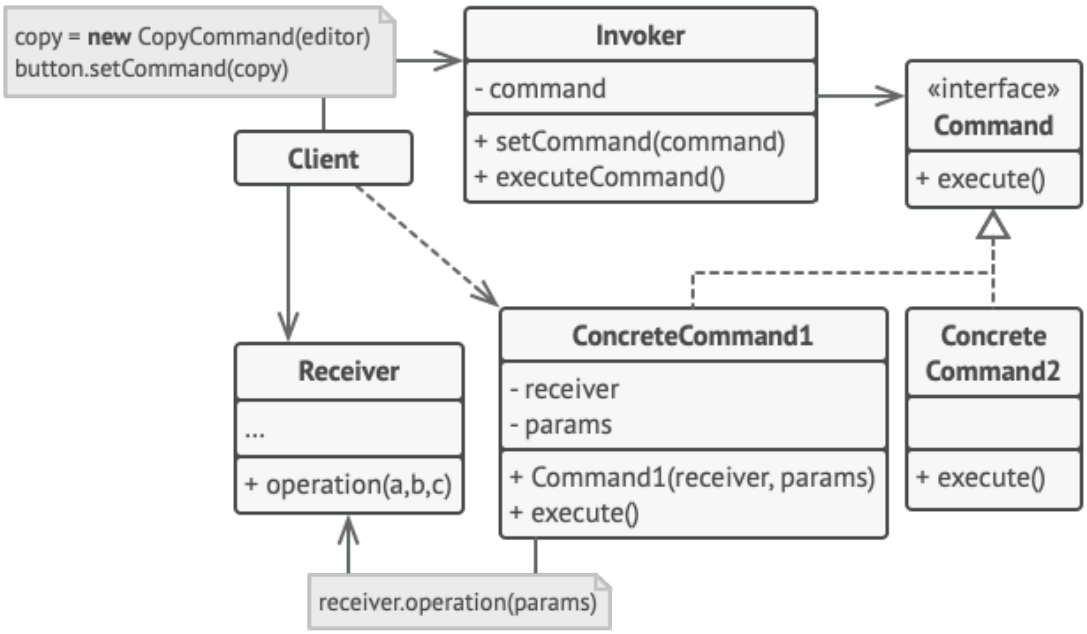

Figura 4.34: Diagramma del pattern Command.

Il pattern **Command** permette di isolare la porzione di codice che effettua un'azione dal codice che richiede la sua esecuzione. L'azione viene quindi incapsulata in un oggetto di tipo `Command`. Lo scopo del pattern è disaccoppiare l'azione di un client dal client stesso, e rendere possibile per una classe eseguire differenti azioni senza necessità di conoscere i dettagli dell'operazione stessa.

Nel progetto il pattern è utilizzato nella parte logica del front end dell'applicativo *SmartLogViewer*, realizzato tramite l'interfaccia `LogManipulator` e le classi `Identity`, `SequenceSearch`, `EventGrouping` e `EventSearch` che la implementano. I metodi così forniti vengono utilizzati da `EventSearchComponent`, `SequenceSearchComponent`, e `TableComponent`.

4.4.2 Decorator

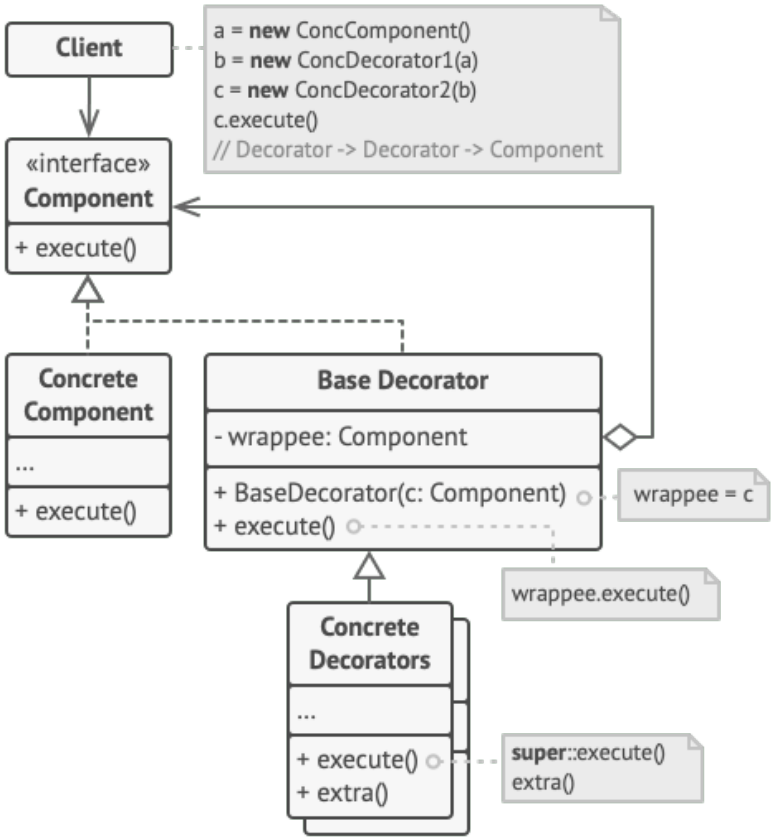

Figura 4.35: Diagramma del pattern Decorator.

Il pattern **Decorator** permette di espandere le funzionalità di un oggetto (classe o metodo) senza modificare l'oggetto stesso, incapsulandolo in un ulteriore oggetto che contiene le funzionalità desiderate.

Nel progetto il pattern è utilizzato nella parte di file upload di entrambe le applicazioni realizzate, implementato tramite l'utilizzo del decorator `@ViewChild` fornito da Angular. Viene inoltre utilizzato, sempre tramite `@ViewChild`, nella parte di front end di *SmartLogViewer*, applicato ad un popup di errore in `SequenceSearch`.

4.4.3 Dependency Injection

Il pattern **Dependency Injection** ha come scopo il passaggio di dipendenze ad un oggetto senza che esso abbia la responsabilità della loro creazione. Questo permette di rendere chiare e apparenti le dipendenze tra i vari elementi.

Nel progetto il pattern è utilizzato nella gestione di vari servizi all'avvio delle due applicazioni *SmartLogViewer* e *SmartLogStatistics*, nello specifico l'avvio di *Swagger* in entrambe le app ed in *SmartLogStatistics* per fornire la classe `SmartLogContext` alle classi che ne hanno bisogno.

4.4.4 Facade

Il pattern **Facade** permette di fornire un'interfaccia semplificata a una libreria, un framework, o un insieme complesso di classi. Rende possibile in questo modo accedere alle funzionalità senza bisogno di interagire in modo diretto con gli elementi a cui è applicato.

Nel progetto, il pattern è utilizzato nella connessione al database tramite la classe `SmartLogContext` (mediante le classi `DbContext` e `DatabaseFacade` dell'Entity Framework) che consente di interagire con il database ed effettuare operazioni su di esso (ad esempio il controllo della presenza di migrazioni e la loro applicazione se necessario).

4.4.5 MVC

Il pattern **MVC** prevede la separazione delle parti di un'applicazione in tre gruppi di componenti:

- **Modello**

  Rappresenta lo stato dell'applicazione e le operazioni che essa deve eseguire, contiene quindi la business logic e qualsiasi implementazione logica per il mantenimento dello stato dell'applicazione.

- **View**

  Presenta il contenuto tramite l'interfaccia utente, contiene il minor quantitativo di logica possibile e la logica contenuta è relativa al presentare il contenuto dell'applicazione.

- **Controller**

  Gestisce le interazioni con l'utente, interagisce con il modello, e sceglie la view da esporre. Controlla e risponde agli input dell'utente relazionandosi con gli altri due elementi.

Le richieste dell'utente passano attraverso un controller il quale svolge le azioni richieste sul model e prende da esso le informazioni richieste. Il controller sceglie la view da mostrate all'utente, e fornisce ad essa i dati necessari dal model. Questa divisione rende più facile scalare il codice e fornisce una separazione chiara dei compiti delle diverse parti del codice.

Nel progetto entrambe le applicazioni sono strutturate seguendo il pattern **MVC**, con classi separate in cartelle `Model` e `Controller` ed un front end realizzato tramite Angular a fare da view.

4.4.6 Observer

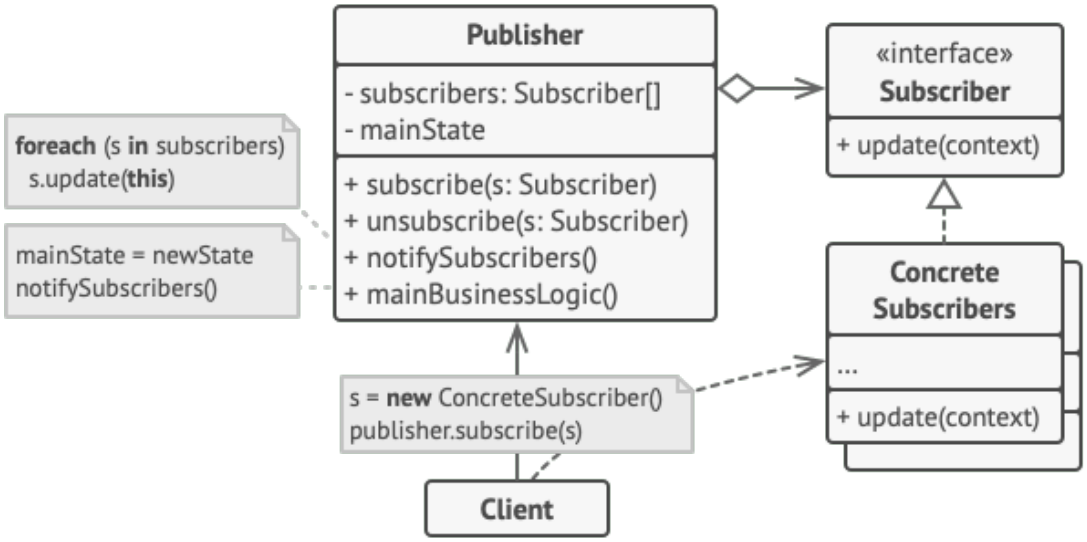

Figura 4.36: Diagramma del pattern Observer.

Il pattern **Observer** prevede la creazione di un meccanismo di notifica che consente ad altri oggetti di "iscriversi" ad un oggetto specifico, e avvisa tali oggetti all'avvenire di un evento (tipicamente un cambiamento di stato) sull'oggetto da essi osservato.

Nel progetto il pattern viene utilizzato nel front end dell'applicazione *SmartLog-Viewer* per osservare il cambiamento dei parametri inseriti dall'utente tramite la classe `LogManipulation`, e aggiornare di conseguenza il grafico visualizzato. Viene similarmente utilizzato nel front end dell'applicazione *SmartLogStatistics* dai component di visualizzazione per osservare i component di ricerca a cui ciascun grafico è associato. Viene inoltre utilizzato da entrambe le applicazioni per gestire la fase di upload dei log.

4.4.7 Proxy

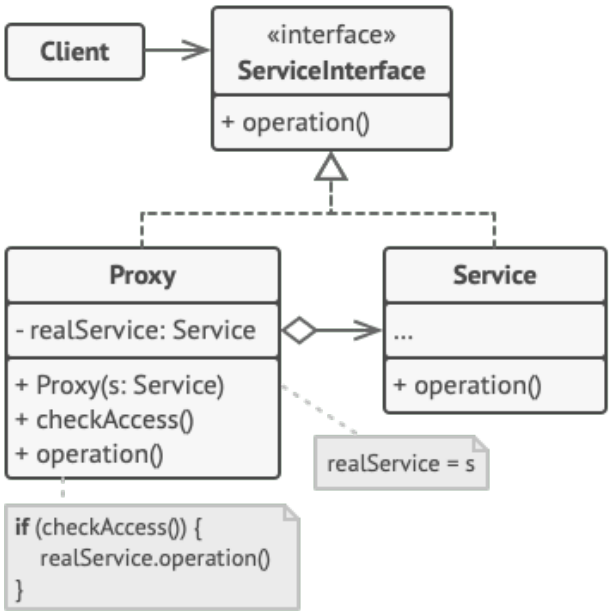

Figura 4.37: Diagramma del pattern Proxy.

Il pattern **Proxy** si basa sul fornire un sostituto o placeholder per un oggetto, e controllare l'accesso all'oggetto originale. Questo permette di intercettare le richieste fatte all'oggetto sostituito e modificarne il contenuto o eseguire operazioni su di esse prima di inoltrare la richiesta all'oggetto. Permette inoltre di eseguire altre operazioni in relazione all'inoltramento delle richieste, prima o dopo averle inoltrate all'oggetto originale.

Nel progetto il pattern è utilizzato per il *lazy loading* di alcuni elementi nella modellazione del database. Viene inoltre utilizzato nei test di unità realizzati tramite il framework **MSTest**.

4.4.8 Singleton

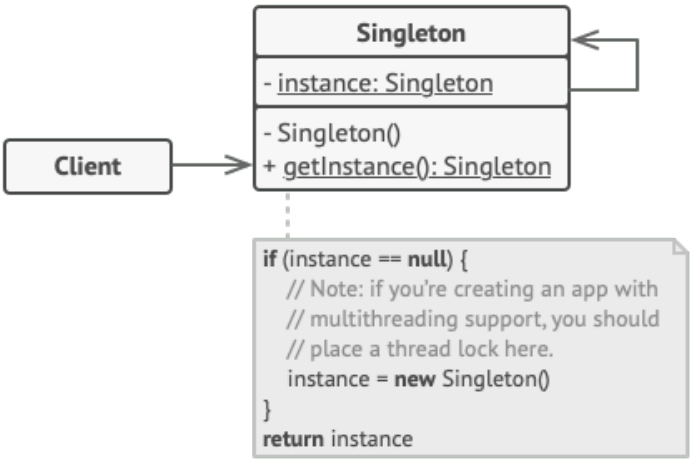

Figura 4.38: Diagramma del pattern Singleton.

Il pattern **Singleton** si basa sull'esistenza di una sola istanza di una data classe, e sul fornire un punto d'accesso globale a tale istanza. Ciò è utile in casi in cui multiple istanze di una stessa classe risultino non desiderabili, come l'accesso ad una risorsa condivisa. Permette inoltre di fornire tale accesso in qualsiasi parte del codice proteggendo comunque l'istanza della classe dall'essere sovrascritta.

Nel progetto il pattern è utilizzato per gestire la connessione e l'accesso al database associato all'applicazione *SmartLogStatistics* tramite la classe `SmartLogContext`, che funge da unico punto di interazione comune con il database.