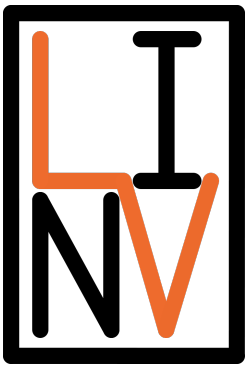

LINV Team

Manuale Sviluppatore

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
	Mauro Carnuccio
	Matteo Cusin
Redattori	Alessandro Baldissera
	Matteo Cusin
	Alessandro Santin

Indice

Elenco delle figure

Elenco delle tabelle

Elenco degli snippet di codice

Registro delle modifiche

Ver.	Data	Autore	Ruolo	Verificatore	Descrizione
1.0	18/06/2023	Matteo Cusin	Responsabile		Approvazione documento
0.7	12/06/2023	Alessandro Baldissera	Sviluppatore	Nicola Ravagnan	Aggiunta sezione di installazione applicativi
0.6	02/06/2023	Matteo Cusin	Sviluppatore	Mauro Carnuccio	Aggiunta descrizione back-end SmartLogStatistics
0.5	21/05/2023	Matteo Cusin	Sviluppatore	Alberto Casado Moreno	Aggiunta descrizione dei problemi noti SmartLogViewer
0.4	16/05/2023	Matteo Cusin	Sviluppatore	Alessandro Baldissera	Aggiunta descrizione front-end SmartLogViewer
0.3	06/05/2023	Alessandro Baldissera	Sviluppatore	Matteo Cusin	Aggiunta descrizione dei punti di estensione back-end SmartLogViewer
0.2	05/05/2023	Alessandro Baldissera	Sviluppatore	Matteo Cusin	Aggiunta descrizione back-end SmartLogViewer
0.1	02/05/2023	Alessandro Santin	Amministratore	Alessandro Baldissera	Stesura preliminare

1. Introduzione

1.1 Scopo del documento

Lo scopo del documento è quello di fornire una linea guida per gli sviluppatori che andranno a manutenere o estendere il prodotto. Di seguito lo sviluppatore troverà tutte le informazioni riguardanti i linguaggi e le tecnologie utilizzati, l'architettura del sistema e le scelte progettuali effettuate nella realizzazione del prodotto.

1.2 Scopo del prodotto

Il capitolato proposto ha come scopo la realizzazione di due applicazioni in grado di visualizzare contenuti e statistiche su file di log riguardanti il funzionamento di prodotti aziendali (gruppi di continuità per impianti industriali). In particolare è richiesto lo sviluppo di un applicativo in grado di visualizzare e permettere l'analisi di singoli file e uno in grado di analizzare e visualizzare statistiche su un insieme di file diversi. Lo scopo del prodotto è fornire strumenti di analisi grafica dei dati disponibili all'azienda per consentire una più facile osservazione degli stessi e una migliore comprensione degli eventi riportati al loro interno e delle possibili correlazioni tra di loro.

1.3 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.

1.4 Riferimenti

1.4.1 Riferimenti normativi

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

1.4.2 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 coinvolte

2.1 Database

2.1.1 PostgreSQL

Per la persistenza dei dati nell'applicativo *SmartLogStatistics*.

2.2 Framework di sviluppo

2.2.1 Angular

Per lo sviluppo dei componenti di interfaccia utente è stato scelto Angular versione 15.2 in quanto fornisce tutti gli strumenti per creare interfacce web dinamiche.

2.2.2 Bootstrap

Il framework Bootstrap è stato impiegato per facilitare lo sviluppo di alcune componenti di interfaccia web (ad esempio le tabelle ed i menù a scomparsa).

2.2.3 MSTest

Framework Microsoft ufficiale per l'implementazione di test dinamici sul software. È stato largamente utilizzato durante lo sviluppo di tutte le componenti server degli applicativi **SmartLogViewer** e **SmartLogStatistics**.

2.2.4 Microsoft .NET 6.0 + ASP.NET Core

Per lo sviluppo dei componenti server di **SmartLogViewer** e **SmartLogStatistics** è stato scelto di utilizzare il framework *Microsoft .NET 6.0 LTS*, con la sua estensione *ASP.NET Core*, per una migliore gestione dei contenuti dinamici da scambiare con il frontend.

2.3 Librerie terze

Nelle seguenti sezioni sono elencate le librerie impiegate per lo sviluppo.

2.3.1 CSVHelper

Utilizzata nella libreria Core per la lettura dei dati CSV dei log. Tale libreria è stata scelta in quanto fornisce un'interfaccia semplice per la lettura di dati formattati in CSV,

accettando come input degli stream, potendo così leggere file di grandi dimensioni ed accettare input direttamente dall'interfaccia HTTP del server senza dover scrivere il file su disco evitando di perdere prestazioni.

2.3.2 Coverlet collector

Plugin del framework di test per la collezione delle statistiche sui test eseguiti. Coverlet permette di generare dei file compatibili con strumenti di analisi e generazione delle metriche di test (ad esempio *CodeCov*, `https://about.codecov.io/`).

2.3.3 Moq

Libreria per la creazione di **mockup** delle classi o interfacce per poter eseguire test di unità in modo efficace ed isolato, svincolandole dal comportamento di altre classi non in esame dalle quali dipendono.

2.3.4 Newtonsoft JSON

Libreria utilizzata da **SmartLogViewer** per la lettura del file di configurazione delle sequenze note. È stato scelto di utilizzare questa libreria in quanto lascia la possibilità di incapsulare gli oggetti JSON in oggetti aventi tipo `dynamic` per effettuare successivamente la validazione delle informazioni fornite dal file.

2.3.5 Npgsql

Libreria utilizzata da *SmartLogStatistics* per il collegamento con il database *PostgreSQL*.

3. Installazione

Nel seguente capitolo sono descritte le procedure di compilazione e installazione del software.

3.1 Download dei sorgenti

I sorgenti di entrambi i software sono contenuti in un repository Git ospitato da GitHub. L'indirizzo del repository è: `https://github.com/linvteam/SmartLogApp`. Per scaricare i sorgenti si può utilizzare il tool Git disponibili per le principali piattaforme, eseguendo il seguente comando:

```
1 git clone https://github.com/linvteam/SmartLogApp.git
```

Oppure, se si vuole utilizzare il protocollo SSH, il comando è il seguente:

```
1 git clone git@github.com:linvteam/SmartLogApp.git
```

3.2 Installazione delle dipendenze

3.2.1 Dipendenze comuni

Per poter compilare correttamente il sorgente è necessario aver installato i tool di `dotnet` e il framework `.NET 6.0 LTS`. Il tool è scaricabile e installabile dal sito ufficile a questo indirizzo `https://dotnet.microsoft.com/en-us/download`, oppure dal gestore pacchetti del proprio sistema operativo, ad esempio `pacman`, `apt` oppure `winget`. I tool di `dotnet` vengono installati automaticamente quando si installa l'IDE **Visual Studio** con l'estensione per `C#` e `dotnet`.

La seconda dipendenza necessaria è `NodeJS 18.16.0 LTS`, anch'esso scaricabile dal sito ufficiale a questo indirizzo `https://nodejs.org/en/download`. Come per la precedente anche questa dipendenza è installabile tramite gestore pacchetti.

3.2.2 Dipendenze SmartLogViewer

L'applicativo **SmartLogViewer** non presenta dipendenze aggiuntive.

3.2.3 Dipendenze SmartLogStatistics

L'applicativo **SmartLogStatistics** necessita anche di `PostgreSQL 15` per poter usufruire del database. La dipendenza è installabile dal sito ufficiale al link `https://www.postgresql.org/download/`, oppure tramite gestore pacchetti del sistema operativo.

3.3 Installazione delle librerie

Le applicazioni dipendeno da alcune librerie, elencate nel documento di ***Specifica Tecnica***, che dovranno essere scaricate prima di poter compilare il software. Sarà necessario aprire un terminale, spostarsi nella cartella del repository ed eseguire il comando:

```
1 dotnet restore
```

3.4 Compilazione dei sorgenti

Per compilare i sorgenti si può procedere tramite un IDE come **Visual Studio** oppure **IntelliJ Rider**, oppure da terminale con il comando:

```
1 dotnet build
```

Questo comando compilerà il software con la configurazione di *Debug*.

3.5 Distribuzione del software

Per la distribuzione del software si esegue il comando:

```
1 dotnet publish -c Release
```

Questo permette di creare il pacchetto definitivo da poter distribuire per l'uso finale. Questo comando compilerà il software con la configurazione di *Release*.

La distribuzione può essere personalizzata tramite alcuni parametri. Di seguito alcuni esempi:

- `--self-contained` permette di integrare il runtime all'interno dell'eseguibile finale per non doverlo installare nella macchina di destinazione;
- `-p:PublishSingleFile=true` permette di integrare nell'eseguibile solo i file necessari all'esecuzione però senza integrare il runtime. Per usare questo parametro è necessario specificare quale progetto compilare tra **SmartLogViewer** e **SmartLogStatistics**;
- `-p:PublishTrimmed=true` permette di ottimizzare le dimensioni dell'eseguibile prodotto, da utilizzare in congiunta con il parametro `--self-contained`;
- `-os <<os>>` permette di specificare il sistema operativo target per la distribuzione, `dotnet` supporta la cross-compilation[1];
- `-a <<arch>>` permette di specificare l'architettura hardware di destinazione, `dotnet` supporta la cross-compilation[2].

Una volta eseguita la compilazione il file che compongono le applicazioni saranno all'interno delle cartelle che seguono il seguente percorso: `{SmartLogViewer|SmartLogStatistics}\bin\Release\net6.0\<<os-arch>>\publish\`

I file contenuti in questa cartella possono essere distribuiti su altre macchine.

[1] Il software è stato testato su Windows, Linux e MacOS

[2] Il software è stato testato per architetture x86-64; sono stati eseguiti dei test preliminari su architetture aarch64 e sistema operativo Linux

3.6 Database di distribuzione per SmartLogStatistics

L'applicativo SmartLogStatistics necessita per l'esecuzione un database PostgreSQL 15 che dovrà essere installato (anche non localmente) e configurato per fornire un utente e un database. L'utente deve avere i permessi per poter manipolare le tabelle del database in quanto l'applicativo stesso si occuperà di generare le tabelle necessarie.

4. Configurazione

4.1 Configurazione di SmartLogViewer

4.1.1 Impostazione delle sequenze note

Sulla cartella di installazione del software è presente un file in formato JSON, chiamato `sequences.json`, che contiene la configurazione di tutte le sequenze note che possono essere identificate nei file di log. Non c'è un limite alle sequenze note che possono essere inserite nel file.
La struttura del file è la seguente:

```
[
    {
        /* Oggetto che descrive la sequenza */
    },
    {
        /* Oggetto che descrive la sequenza */
    },
    ...
]
```

Listing 4.1: Struttura del file che contiene le sequenze note

L'oggetto che descrive la sequenza ha il seguente formato:

```
{
    "Name": string,
    "StartEvents": [
        {
            "Code": string,
            "Status": boolean
        }
    ],
    "StartEventsAvailableSubUnits": [ number ],
    "EndEvents": [
        {
            "Code": string,
            "Status": boolean
        }
    ],
    "EndEventsAvailableSubUnits": [ number ],
    "MaxDuration": number
}
```

Listing 4.2: Struttura dell'oggetto che descrive una sequenza nota

In questo oggetto devono essere presenti i seguenti campi:

- `Name`: indica il nome della sequenza nota;
- `StartEvents`: una lista che contiene tutti i code degli eventi e relativi stati per considerare iniziata una sequenza;
- `StartEventsAvailableSubUnits`: una lista di numeri che indica le subUnit sulle quali possono avvenire gli eventi di avvio sequenza;
- `EndEvents`: una lista che contiene tutti i code degli eventi e relativi stati per considerare terminata una lista;
- `EndEventsAvailableSubUnits`: una lista di numeri che indica le subUnit sulle quali possono avvenire gli eventi di fine sequenza;
- `MaxDuration`: il tempo in millisecondi entro il quale una sequenza deve terminare per essere considerata valida.

4.1.2 Modifica delle sequenze note

Per modificare la configurazione delle sequenze note occorre modificare direttamente il contenuto del file `sequences.json`:

- **Aggiunta di una sequenza**: per aggiungere una sequenza occorre inserire, come mostrato dalla figura relativa alla struttura del file delle sequenze, un oggetto che segua la struttura mostrata dalla relativa figura;
- **Rimozione di una sequenza**: per rimuovere una sequenza occorre eliminare per intero un oggetto rappresentante una sequenza e le parentesi graffe che lo circondano (compresa la virgola adiacente all'ultima parentesi, se presente);
- **Modifica di una sequenza**: per modificare una sequenza occorre modificare il valore di uno o più campi all'interno dell'oggetto rappresentante la sequenza scelta, rispettando la notazione ed il tipo di dati richiesti.

4.2 Configurazione di SmartLogStatistics

L'applicativo SmartLogStatistics necessita della sola configurazione delle impostazioni di connessione al database. Nella cartella `SmartLogStatistics` è presente un file chiamato `appsettings.json` che contiene la configurazione di avvio dell'applicazione. La configurazione di default è la seguente:

```
1 {
2   "Logging": {
3     "LogLevel": {
4       "Default": "Information",
5       "Microsoft.AspNetCore": "Warning"
6     }
7   },
8   "AllowedHosts": "*",
```

```
9   "ConnectionStrings": {
10    "SmartLogContext":
        ↪ "Host=localhost;Database=Statistics;Username=Utente;
        ↪ Password=Password"
11  }
12 }
```

Listing 4.3: File di configurazione di SmartLogStatistics

Alla riga 10 è presente la stringa di connessione al database che può essere personalizzata. Consigliamo, per motivi di sicurezza, di cambiare username e password di accesso al database.

5. Strumenti di analisi e integrazione del codice

6. Architettura

6.1 Libreria Core

6.1.1 Introduzione

La libreria **Core** contiene le funzionalità condivise tra le applicazioni **SmartLogViewer** e **SmartLogStatistics**.

6.1.2 Dipendenze

La libreria **Core** dipende dai seguenti pacchetti:

- CSVHelper (versione 30.0.1).

6.1.3 Principali componenti

- `RawLogRow` e `LogRow`: strutture di supporto per i dati di un record del log; le annotazioni permettono di indicare alla libreria `CSVHelper` il nome esatto dei campi;
- `Header`: struttura di supporto per i dati dell'header del file di log;
- `INIFile`: struttura di supporto per i nomi dei file INI presenti nell'header del file di log;
- `Log`: struttura di supporto che aggrega header e dati del file di log;
- `HeaderParser`: classe che si occupa di leggere l'header del file di log a partire da un oggetto di tipo `TextReader`;
- `DataParser`: classe che si occupa di leggere la tabella CSV interfacciandosi con la libreria `CSVHelper`.

6.1.4 Test del sorgente

I test delle componenti software sono contenuti nel progetto chiamato **CoreTests**.

6.2 Back-end SmartLogViewer

6.2.1 Introduzione

Il back-end dell'applicativo **SmartLogViewer** è composto da un server HTTP che espone delle API HTTP RESTful che consentono di:

- Leggere un file di log in formato CSV avente un header custom;
- Leggere un file contenente le routine note dei macchinari;
- Distribuire i file statici dell'interfaccia front-end.

6.2.2 Dipendenze

Le dipendenze del back-end di **SmartLogViewer** sono:

- Newtonsoft JSON;
- Libreria Core.

6.2.3 Principali componenti

Le principali componenti del back-end di **SmartLogViewer** sono:

- `Sequence` e `Event`: strutture di supporto che contengono tutte le informazioni relative ad una sequenza di eventi nota;
- `SequencesManagerJson`: classe che si occupa di leggere il file delle sequenze (avente formato JSON) e renderle disponibili alle altre componenti; lo `StreamReader` del file delle sequenze viene iniettato tramite la classe `SequenceFileReader`;
- `ParseController`: classe che si occupa di esporre le API di lettura del file di log;
- `EventSequenceController`: classe che si occupa di esporre le API per ottenere le informazioni sulle sequenze note.

6.2.4 Test del sorgente

I test delle componenti software sono contenuti nel progetto chiamato **SmartLogViewer-Tests**.

6.3 Front-end SmartLogViewer

6.3.1 Introduzione

Il front-end dell'applicativo **SmartLogViewer** è composto da una serie di **components**$_G$ e **services**$_G$ dediti a:

- Selezionare un file di log in formato CSV da un filesystem locale;
- Ottenere una rappresentazione del file di log tramite chiamata HTTP POST al back-end dell'applicativo;
- Ottenere una rappresentazione delle sequenze di eventi disponibili per la ricerca tramite chiamata HTTP GET al back-end dell'applicativo;
- Visualizzare le informazioni del file di log selezionato tramite tabelle e grafici;
- Filtrare gli eventi visualizzati;
- Ricercare sequenze di eventi note.

6.3.2 Dipendenze

Le dipendenze del front-end di **SmartLogViewer** sono:

- Back-end SmartLogViewer;
- Dipendenze npm (ad esempio Bootstrap).

6.3.3 Principali componenti

Le principali componenti del front-end di **SmartLogViewer** sono:

- Component$_G$ `File-Upload` e service$_G$ `File-Upload`: si occupano di gestire il caricamento del file di log ed ottenere risposta dal back-end dell'applicativo per quanto riguarda le operazioni di parsing dello stesso;
- Service$_G$ `Log`: gestisce l'occorrenza del log utilizzata in un certo istante temporale e fornisce i suoi dati alle classi che li richiedono;
- Component$_G$ `Table`: gestisce la visualizzazione tabellare degli eventi binari del file di log, anche in forma raggruppata;
- Component$_G$ `Chart`: gestisce la visualizzazione tramite grafico di attivazione degli eventi binari del file di log;
- Service$_G$ `Sequence-Fetch`: si occupa di ottenere le sequenze di eventi disponibili per la ricerca dal back-end dell'applicativo.

6.3.4 Test del sorgente

I test delle componenti software (file `.ts`) sono contenuti in file omonimi di estensione `.spec.ts`.

6.4 Back-end SmartLogStatistics

6.4.1 Introduzione

Il back-end dell'applicativo **SmartLogStatistics** è composto da un server HTTP che espone delle API HTTP RESTful che consentono di:

- Leggere uno o più file di log in formato CSV aventi un header custom;
- Salvare i file di log letti in un database *PostgreSQL*;
- Ottenere informazioni e statistiche sui file di log caricati nel database.

6.4.2 Dipendenze

Le dipendenze del back-end di **SmartLogStatistics** sono:

- Libreria Core;
- Libreria Npgsql.

6.4.3 Principali componenti

Le principali componenti del back-end di **SmartLogStatistics** sono:

- Classi di comunicazione col database:
  - `Event`, `Log`, `LogFile` e `Firmware`: modellano la struttura della base di dati;
  - `SmartLogContext`: contengono tutte le informazioni per la creazione, correzione e comunicazione con la base di dati su cui si andrà ad operare.
- `UploadController`: classe che si occupa di esporre le API di conversione dei file di log;
- `StatisticsController`: classe che si occupa di esporre le API per ottenere statistiche su un sottoinsieme desiderato dei file di log salvati;
- `DataController`: classe che si occupa di esporre le API per ottenere informazioni da un sottoinsieme desiderato dei file di log salvati (come, ad esempio, la frequenza di occorrenza di eventi raggruppati per uno o più campi dati scelti dall'utente);
- `InfoController`: classe che si occupa di esporre le API per ottenere metadati sull'insieme dei file di log salvati.

6.4.4 Test del sorgente

I test delle componenti software sono contenuti nel progetto chiamato **SmartLogStatisticsTests**.

6.5 Persistence layer SmartLogStatistics

6.5.1 Principali componenti

Lo strato di persistenza è costituito da una base di dati PostgreSQL dotata delle seguenti tabelle:

- **Event**: modella le varie tipologie di eventi, memorizzandone il codice, la descrizione ed il colore (per una possibile rappresentazione delle occorrenze di quel tipo di evento);
- **Log**: modella le occorrenze degli eventi, memorizzandone il file di appartenenza, la linea in cui si trova l'occorrenza nel file, la data/ora di avvenimento, la Unit, la SubUnit, il valore booleano ed il codice dell'evento;
- **File**: modella i file contenenti le occorrenze di eventi, memorizzandone il nome, il campo *PCDateTime* ed il campo *UPSDateTime*;
- **Firmware**: modella i firmware utilizzati dai macchinari, memorizzandone Unit, SubUnit e nome.

7. Punti di estensione

Nella seguente sezione sono descritti i principali punti di estensione delle applicazini, separate per server di back-end e interfaccia front-end.

7.1 Introduzione generale

Di seguito sono riportate alcune indicazioni generali sulle possibili modalità di estensione delle funzionalità del software.

7.1.1 Uso delle annotazioni

Nei controller è importante utilizzare le annotazioni nel codice per indicare al framework alcune caratteristiche del comportamento dei metodi.

7.1.1.1 Principali annotazioni usate nel codice

Le possibili annotazioni per i metodi sono:

- `HttpGet`: per indicare che il metodo risponde a chiamte di tipo **GET**;
- `HttpPost`: per indicare che il metodo risponde a chiamte di tipo **POST**;
- `Produces("application/json")`: per indicare il tipo dell'output della chiamata, segue le convenzioni dell'header HTTP `Content-Type`;
- `ProducesResponseType(StatusCodes.Status500InternalServerError)`: per indicare un possibile codice HTTP che produce la chiamata, ci possono essere più annotazioni di questo tipo con codici diversi;
- `ProducesResponseType(typeof(Log), StatusCodes.Status200OK)`: per indicare un possibile codice HTTP prodotto e il tipo di oggetto ritornato;
- `Route("api/{controller}")`: per indicare l'endpoint in cui tale controller rimarrà in ascolto, indicando `{controller}` si indica al framework che il controller risponderà al percorso composto dal nome del controller senza il suffisso `Controller` (può essere sostituito da un nome qualsiasi).

Si consiglia di consultare la documentazione ufficiale per un elenco esaustivo delle possibili annotazioni di metodi e classi.

```
1 [HttpPost]
2 [ProducesResponseType(typeof(Log),
    ↪ StatusCodes.Status201Created)]
```

```
3 [ProducesResponseType(typeof(ParseError),
    ↪ StatusCodes.Status400BadRequest)]
4 [Produces("application/json")]
5 public IActionResult Upload(IFormFile file)
6 {
7     ...
8 }
```

Listing 7.1: Esempio di metodo arricchito con annotazioni

7.1.1.2 Perchè utilizzare le annotazioni

L'uso delle annotazioni risulta essere molto utile in quanto permette di ottenere i seguenti benefici:

- Codice sorgente con maggiore documentazione;
- Possibilità di utilizzare il plug-in **Swagger** per visualizzare ed esportare la documentazione in formato OpenAPI, oltre ad avere la possibilità di testare l'interfaccia senza l'ausilio di strumenti esterni.

7.1.2 Commenti XML

Tutto il software scritto in C# supporta la documentazione direttamente sul sorgente tramite commenti contenenti informazioni codificate tramite XML. Per attivare questo tipo di documentazione basta commentare con `///` prima della definizione di un metodo o prima delle eventuali annotazioni. Utilizzando un IDE quale Visual Studio o IntelliJ Rider, la struttura XML verrà inserita automaticamente e sarà sufficiente compilarla con la documentazione necessaria. Documentare i metodi e le classi con questa tecnica permette anche agli IDE di estrarre automaticamente le informazioni riguardanti classi, interfacce, metodi e proprietà per renderle poi disponibili al programmatore.

7.1.2.1 Tag XML

I principali tag XML per la documentazione sono:

- `<summary></summary>` specifica una descrizione di ciò che si vuole documentare;
- `<param name="param_name"></param>` descrive il parametro `param_name` del metodo;
- `<returns></returns>` descrive cosa viene ritornato dal metodo;
- `<exception cref="exception_type"></exception>` descrive una possibile eccezione lanciata dal metodo, il parametro `cref` specifica il tipo dell'eccezione lanciata; ci possono essere tanti tag `exception` quanti tipi di eccezione vengono lanciati dal metodo;
- `<response code="200"></response>` caratteristica dei metodi che gestiscono chiamate HTTP, questo tag descrive il contenuto ritornato dal metodo rispetto ad un determinato codice di stato HTTP specificato nel parametro `code`.

Anche in questo caso si consiglia di consultare la documentazione ufficiale per maggiori informazioni sui commenti XML.

```
1  /// <summary>
2  /// Ritorna un JSON che rappresenta il file di log in
       ↪ ingresso (dopo essere stato filtrato)
3  /// </summary>
4  /// <param name="file">File di cui deve essere eseguito il
       ↪ parsing</param>
5  /// <returns>Esito della chiamata POST, o un file JSON che
       ↪ rappresenta il file di log oppure un'eccezione dovuta
       ↪ al parsing del file</returns>
6  /// <response code="201">Ritorna il file
       ↪ convertito</response>
7  /// <response code="400">Ritornato se viene scatenato un
       ↪ errore nella conversione</response>
8  public IActionResult Upload(IFormFile file)
9  {
10     ...
11 }
```

Listing 7.2: Esempio di metodo documentato con commenti XML

7.1.2.2 Perchè utilizzare i commenti XML

L'uso dei commenti può tornare molto utile perchè:

- Permettono di scrivere la documentazione direttamente all'interno del codice sorgente e non tramite strumenti esterni;
- Permette ai principali IDE di estrarre e fornire al programmatore la documentazione dei metodi, classi, interfacce, etc;
- Permette a plug-in come **Swagger** di estrarre informazioni aggiuntive da fornire al programmatore nell'interfaccia di consultazione delle documentazione delle API RESTful;
- Permette di utilizzare strumenti esterni per generare documenti che raccolgono l'intera documentazione del software automaticamente (ad esempio **doxygen**).

7.1.3 Commenti TypeScript

Tutto il software scritto in TypeScript supporta la documentazione direttamente sul sorgente tramite commenti contenenti informazioni codificate tramite specifica TSDoc (`https://tsdoc.org/`). Per attivare questo tipo di documentazione basta iniziare un commento con `/**` (e concluderlo con `*/`) prima della definizione di un metodo. Utilizzando un IDE quale Visual Studio o IntelliJ Rider, la struttura verrà inserita automaticamente e sarà sufficiente compilarla con la documentazione necessaria. Documentare i metodi e le classi con questa tecnica permette anche agli IDE di estrarre automaticamente le informazioni riguardanti classi, interfacce, metodi e proprietà per renderle poi disponibili al programmatore.

7.1.3.1 Annotazioni

Le principali annotazioni per la documentazione sono:

- `@param param\_name` descrive il parametro `param_name` del metodo;
- `@returns` descrive l'oggetto ritornato dal metodo.

Anche in questo caso si consiglia di consultare la documentazione ufficiale per maggiori informazioni sui commenti (TSDoc è una proposta di standardizzazione al momento di).

```
1     /** Trova gli eventi del file di log uguali ad una serie
          ↪ di eventi passati
2      * @param log il file di log attuale
3      * @param EventsToFind l'insieme di eventi da trovare
4      * @param acceptedSubUnits le subUnit in cui gli eventi
          ↪ possono essere avvenuti
5      * @returns Ritorno gli eventi dei tipi richiesti che
          ↪ riguardano le subUnit indicate
6      */
7     private findEvents(log : Log, EventsToFind : Event[],
        ↪ acceptedSubUnits: number[]) : LogRow[] {
8           ...
9     }
```

Listing 7.3: Esempio di metodo documentato con commenti TSDoc

7.1.3.2 Perchè utilizzare i commenti TSDoc

L'uso dei commenti può tornare molto utile perchè:

- Permettono di scrivere la documentazione direttamente all'interno del codice sorgente e non tramite strumenti esterni;
- Permette ai principali IDE di estrarre e fornire al programmatore la documentazione dei metodi, classi, interfacce, etc;
- Permette di utilizzare strumenti esterni per generare documenti che raccolgono l'intera documentazione del software automaticamente (ad esempio il pacchetto npm **Compodoc**).

7.2 Libreria Core

Estendere la libreria core può essere utile quando è necessario aggiungere delle funzionalità condivise tra entrambi i server back-end degli applicativi **SmartLogViewer** e **SmartLogStatistics**.

7.3 Back-end

Nella sezione seguente sono descritti i principali punti di estensione delle funzionalità di back-end dell'applicativo **SmartLogViewer**.

7.3.1 Aggiunta di nuove API

7.3.1.1 Aggiunta di una nuova categoria di API

Al backend possono essere aggiunte nuove API che non sono correlate a quelle già esistenti. Per farlo, è necessario aggiungere una nuova classe così strutturata:

```
1  using Microsoft.AspNetCore.Mvc;
2
3  namespace SmartLogViewer.Controllers
4  {
5      [Route("api/{controller}")]
6      [ApiController]
7      public class CustomController : ControllerBase {
8
9          [HttpGet]
10         public IActionResult Get() {
11             ...
12         }
13     }
14 }
```

Listing 7.4: Struttura di base di un controller

Nel costrutture della classe è possibile iniettare le dipendenze che dovranno poi essere aggiunte al dependency injector.

7.3.1.2 Aggiunta di API ad un categoria già esistente

È possibile aggiungere delle nuove API ad una categoria esistente andando ad aggiungere dei nuovi metodi ad uno dei controller già esistenti. Per farlo sarà necessario aggiungere un nuovo metodo con le annotazioni che descrivono il metodo HTTP al quale risponderà e l'eventuale route a cui deve rispondere.

```
1  [HttpGet]
2  [Route("route")] // Opzionale
3  public IActionResult NuovoMetodoDelController() {
4      ...
5  }
```

Listing 7.5: Esempio di metodo per implementare una nuova API

7.3.2 Aggiunta di componenti del modello

Per aggiungere componenti nel modello è necessario aggiungere classi o interfacce nella cartella `Model`. Nel costruttore della classe si possono aggiungere le dipendenze di tale classe che verranno iniettate dal dependency injector. Per poter utilizzare la classe come dipendenza per altre, sarà necessario aggiungerla al dependency injector. Per fare ciò si deve aggiungere al file `Program.cs` la seguente linea:

```
1  builder.Services.AddSingleton<T>();
2  builder.Services.AddSingleton<IT, T>();
```

Listing 7.6: Aggiunta di classi del modello al dependency injector

Nella prima riga si indica che la classe di tipo `T` verrà aggiunta a tutte le classi istanziate dal dependency injector che richiedono un attributo di tipo `T` sul costruttore. Nella seconda riga si indica che la classe di tipo `T` verrà aggiunta a tutte le classi istanziate dal dependency injector che richiedono un attributo sul costruttore che implementa l'interfaccia `IT`. Nel secondo caso, le classi verranno istanziate tramite il pattern **singleton**.

7.3.2.1 Cambio del formato del file delle sequenze note

È possibile cambiare il formato del file delle sequenze note da JSON ad un altro formato. Per farlo sarà necessario scrivere una classe che implementi l'interfaccia `SequencesManager`. Una volta implementata basterà indicare sul file `Program.cs` la classe concreta da utilizzare nella riga:

```
1 builder.Services.AddSingleton<SequencesManager,
    ↪ SequencesManagerJson>();
```

Listing 7.7: Cambio della classe di lettura del file delle sequenze

Dove si sostituisce la classe `SequencesManageJson` con la classe appena creata.

7.4 Front-end

Nella sezione seguente sono descritti i principali punti di estensione delle funzionalità di front-end dell'applicativo **SmartLogViewer**.

7.4.1 Aggiunta di nuovi components

Per aggiungere un nuovo component$_G$ è necessario aggiungere una cartella omonima all'interno del folder `components`; tale cartella deve contenere i file:

- `<nome-component>.html`: contiene la struttura HTML del nuovo componente;
- `<nome-component>.css`: contiene lo stile del nuovo componente;
- `<nome-component>.ts`: contiene la logica di base del nuovo componente;
- `<nome-component>.spec.ts`: contiene i test della classe implementata nel file `<nome-component>.ts` del nuovo componente;

Il nuovo component deve essere registrato all'interno del file `app.module`, nell'array `declarations`. Tutte queste operazioni possono essere svolte automaticamente usando il seguente comando da terminale:

```
1     ng generate component components/<nome-component>
```

7.4.2 Aggiunta di nuovi services

Per aggiungere un nuovo service$_G$ è necessario aggiungere una cartella omonima all'interno del folder `services`; tale cartella deve contenere i file:

- `<nome-service>.ts`: contiene la logica di base del nuovo service;

- `<nome-service>.spec.ts`: contiene i test della classe implementata nel file `<nome-service>.ts` del nuovo service.

La creazione dei file può essere eseguita automaticamente usando il seguente comando da terminale:

```
1 ng generate service service/<nome-service>
```

8. Problematiche note

Nella seguente sezione sono descritti i principali problemi noti degli applicativi. Nonostante questi problemi, gli applicativi possono sempre essere riportati in uno stato consistente tramite interfaccia grafica.

8.1 SmartLogViewer

- Ricaricare l'applicativo, mentre si stanno visualizzando le pagine web dedicate alla visualizzazione dei dati, restituisce una pagina vuota (comprendente la barra di navigazione con la voce **Carica un File**); occorre eseguire un redirect verso la pagina di caricamento del file di log (path `/file-upload`);
- Non è stato eseguito un controllo sul dominio dei valori dei campi *Unit* e *SubUnit* degli eventi del file di log, per quanto riguarda la possibilità di avere valori negativi.