Università degli Studi di Padova

Dipartimento di Matematica "Tullio Levi-Civita"

Corso di Laurea in Informatica

Sviluppo di un applicativo per lo scambio di dati sul traffico veicolare tra le piattaforme TMacs e Enel X

Tesi di laurea triennale

Relatore

Prof. Luigi De Giovanni

Laureando

Marco Bustaffa
Matr. 1226301

Anno Accademico 2021-2022

Coloro che sognano di giorno sanno molte cose che sfuggono a chi sogna soltanto di notte.

— Edgar Allan Poe

Dedicato ad Enrico per il sostegno e i preziosi consigli.

Sommario

Il presente documento descrive il lavoro svolto durante il periodo di stage, della durata di circa trecento ore, dal laureando Marco Bustaffa presso l'azienda TECSEN S.r.l nel periodo che va dal 12/07/2022 al 6/09/2022.

Lo scopo dello stage è la realizzazione di un applicativo center to center per lo scambio di dati sul traffico veicolare tra le piattaforme TMacs ed Enel X.

TMacs è l'applicativo sviluppato da Tecsen per il controllo e il monitoraggio remoto delle intersezioni semaforiche, mentre la piattaforma di Enel X si pone l'obiettivo di raccogliere i dati sul traffico per fornire ai propri utenti informazioni utili sui flussi veicolari.

In questo documento viene descritta l'organizzazione del suddetto stage, la fase di analisi del problema e la conseguente progettazione. In particolare, viene descritta la necessità di produrre un applicativo stabile e robusto, ma altrettanto configurabile, e come esso comunichi con la piattaforma di Enel X.

Convenzioni tipografiche

Riguardo la stesura del testo, relativamente al documento sono state adottate le seguenti convenzioni tipografiche:

- gli acronimi, le abbreviazioni e i termini ambigui o di uso non comune menzionati vengono definiti nel glossario, situato alla fine del presente documento;
- per la prima occorrenza dei termini riportati nel glossario viene utilizzata la seguente nomenclatura: *parola*[g].

Ringraziamenti

Innanzitutto, vorrei esprimere la mia gratitudine al Prof. Luigi De Giovanni, relatore della mia tesi, per l'aiuto e il sostegno fornitomi durante la stesura del lavoro.

Desidero ringraziare con affetto i miei genitori per il sostegno, il grande aiuto e per essermi stati vicini in ogni momento durante gli anni di studio.

Ho desiderio di ringraziare l'Ingegnere Brunello Andrea, tutor aziendale, per la sua grande disponibilità dimostrata nel periodo di stage.

Padova, Dicembre 2022 Marco Bustaffa

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

In questo capitolo viene fornita una breve introduzione al progetto e alla soluzione realizzata.

1.1 L'azienda

TEC Systems Engineering S.r.l. [17] (vedi Figura 1.1) nasce nel 2007, si sviluppa e cresce grazie alla Partnership e al continuo scambio con il gruppo La Semaforica [9] leader nazionale nei sistemi di regolazione semaforica e del traffico. Da qui prende vita la collaborazione esclusiva per lo sviluppo di ITS[g] (Intelligent Transport System) ovvero sistemi che attraverso l'informatica, l'elettronica e l'ingegneria dei trasporti permettono di monitorare ed interagire con impianti semaforici remoti.

Figura 1.1: logo Tecsen S.r.l.

1.2 Introduzione al progetto

La proposta di stage era quella di soddisfare le richieste di Enel X [1], ovvero il reparto smart di Enel, azienda che si occupa di garantire servizi all'avanguardia e sempre legati alle tematiche di energia e urbanità.

Enel X infatti, con l'intenzione di produrre un applicativo di data visualization dei dati traffico italiani, ha contattato Tecsen S.r.l. richiedendo a quest'ultima la possibilità di ricevere ed elaborare i dati raccolti dagli impianti semaforici centralizzati.

A quel punto, dopo processo di contrattazione e definizione delle specifiche e dei requisiti principali, Tecsen ha avviato il progetto.

Il progetto di stage proposto dall'azienda prevede lo sviluppo di un applicativo software in grado di raccogliere, elaborare e trasmettere i dati traffico verso gli endpoint[g] disposti da Enel X.

La soluzione software presentata da Tecsen [17] non predispone un frontend[g], questo perchè non è necessario essendo un'applicativo che non richiede l'interfacciamento con un utente, è composta quindi da solo il backend[g] in Java.

1.3 Soluzione scelta

L'architettura individuata per l'applicativo è basata sul paradigma REST[g] (Representational state transfer), questo per rispettare gli standard moderni e fornire un servizio leggero e altamente scalabile.

Il backend raccoglie i dati attraverso delle query indirizzate ai database disposti dall'azienda nei loro quattro server, li elabora secondo le specifiche concordate con il cliente e li inoltra componendo delle richieste cURL[g] verso i loro endpoint.

Per l'implementazione dell'applicativo, è stato utilizzato Java nella sua versione nativa, questo perchè l'azienda non ha riscontrato la necessità di appoggiarsi a framework[g] di terze parti.

Questa scelta è stata dettata da diverse motivazioni, in particolare l'idea era quella di consegnare l'applicativo al termine dello stage e quindi di non investire ulteriore tempo nell'apprendimento di un framework che poi non avrebbe trovato spazio nell'azienda.

1.4 Principali problematiche

Durante lo svolgimento dello stage si sono presentate alcune criticità, ovvero:

- il cliente, nonostante le specifiche già concordate con Tecsen [17] prima del mio arrivo, ha richiesto diverse modifiche in corso d'opera, ma che fortunatamente non hanno causato variazioni importanti alle attività e alla pianificazione dello stage;
- l'applicativo non poteva essere testato su un server reale e con dati aggiornati durante il suo sviluppo, questo perchè correre il rischio di mandare offline il sever avrebbe comportato grosse problematiche all'azienda;
- il cliente che doveva predisporre gli endpoint per far si che potessimo inviargli i dati, ha avuto alcuni rallentamenti interni, che di conseguenza si sono riflessi sull'organizzazione delle tempistiche del progetto.

1.5 Organizzazione del testo

Il Capitolo 2 descrive gli strumenti e le tecnologie che sono state utilizzate.

Il Capitolo 3 descrive gli obiettivi, il metodo di lavoro e la struttura dello stage.

Il Capitolo 4 approfondisce gli aspetti progettuali e di codifica dell'applicativo.

Il Capitolo 5.1 descrive il processo di validazione e verifica.

Il Capitolo 6 trae le conclusioni finali sull'esperienza di stage.

Capitolo 2

Strumenti e ambiente di sviluppo

In questa sezione verranno descritti i Sistemi Operativi, i software preinstallati e il database utilizzati in azienda per la corretta realizzazione del progetto di stage.

2.1 Il sistema operativo

I computer presenti in azienda utilizzavano due diverse tipologie di sistema operativo: Linux Ubuntu 22.10 [18] e Windows 10 [20]. Per comodità e familiarità con l'ambiente, ho scelto di utilizzare Windows 10, ma i software che sono stati utilizzati sono pienamente compatibili con entrambi i sistemi operativi.

2.2 NetBeans

NetBeans [13] (vedi Figura 2.1) è un ambiente di sviluppo integrato multi-linguaggio, nato nel giugno 2000 e scritto interamente in Java [7], scelto dalla Oracle Corporation come IDE (Integrated Development Environment) ufficiale da contrapporre al più diffuso Eclipse.
L'azienda ha deciso di adottare NetBeans come ambiente di sviluppo integrato di default in quanto di semplice utilizzo ma completo nelle sue funzionalità.
L'alternativa sarebbe potuta essere Visual Studio Code, un editor di testo prodotto da Microsoft, capace attraverso delle estensioni aggiuntive, di offrire tutte le funzionalità di un ambiente di sviluppo integrato più classico.
Per semplicità e facilità di setup, ho scelto di utilizzare NetBeans come mio ambiente di sviluppo integrato principale.

Figura 2.1: logo Netbeans

2.3 Git/GitAhead

Git [2] (vedi Figura 2.2) è un sistema di versionamento che permette la collaborazione di diversi sviluppatori allo stesso progetto.
Anche l'azienda ha deciso di versionare i propri progetti con questo sistema, in quanto efficace ed efficiente.
L'azienda non utilizza però piattaforme di supporto a Git come GitHub [5], ha predisposto invece un server interno in cui è stato installato Git. Tutti i progetti vengono versionati su questo server, seguendo delle specifiche regole imposte dall'azienda, ad esempio:

* quando si inizializza un nuovo progetto Git[2], assicurarsi che il branch principale sia *main* e non *master*.
* assicurasi che sia presente il file *.gitignore* in tutti i progetti.
* quando si apportano delle modifiche ad un progetto lavorare sempre in un proprio branch dedicato.

In azienda per facilitare l'utilizzo del sistema di vesrionamento Git è stato indtrodotto GitAhead [3] (vedi Figura 2.3). GitAhead è un'interfaccia grafica di Git per i sistemi Windows, Linux e macOS e dispone di tutte le funzionalità che Git [2] normalmente offre ma introducendo un'interfaccia chiara e semplice da apprendere.

Figura 2.2: logo Git

Figura 2.3: logo GitAhead

2.4 Payara

Payara [15] (vedi Figura 2.4) server è un application server open source derivato dal più conosciuto GlassFish Server, venne creato nel 2014 dopo che Oracle annunciò lo stop al supporto di GlassFish.
In'azienda, Payara [15] viene utilizzato come application server principale, e tutti gli applicativi che Tecsen [17] offre spesso sono eseguiti al suo interno.

Figura 2.4: logo Payara

2.5 Java

Java [7] (vedi Figura 2.5) è un linguaggio di programmazione ad alto livello, orientato agli oggetti e a tipizzazione statica, che si appoggia sull'omonima piattaforma software di esecuzione (JVM[g], Java Virtual Machine), specificamente progettato per essere il più possibile indipendente dalla piattaforma hardware di esecuzione.
In azienda, tutti gli applicativi desktop e tutti i protocolli di comunicazione con i regolatori semaforici sono scritti in Java, la motivazione principale per la quale Java è così largamente diffuso in azienda, e più in generale nel mondo, è proprio perchè il funzionamento dell'applicativo non dipende dal tipo di hardware sul quale esegue.

Figura 2.5: logo Java

2.6 Gson

Gson [6] (vedi Figura 2.6) è una libreria Java [7] open source, di proprietà di Google, per serializzare e deserializzare oggetti Java su JSON[g]. È una libreria molto efficace e di semplice utilizzo e di conseguenza non è stato difficile apprenderne il funzionamento. Vengono forniti due metodi principali, il primo chiamato *toJson()* per serializzare oggetti Java [7], o stringhe di testo, in file di tipo JSON, il secondo chiamato *fromJson()* per eseguire l'operazione di deserializzazione in oggetti Java [7] e stringhe di testo.

Figura 2.6: logo Gson

2.7 Redmine

Redmine [16] (vedi Figura 2.7) è un software gratuito e open source, per la pianificazione di progetti e per il tracciamento delle segnalazioni di bug tramite interfaccia web.
Redmine è stato scelto dall'azienda in quanto completo di tutte le funzionalità necessarie al tracciamento e alla pianificazione delle attività e anche per il fatto che è totalmente open source.
Ogni attività su Redmine non solo era accompagnata da una data di fine e da una data d'inizio ma anche da altrettante informazioni utili. Alcune tra queste sono ad esempio: la priorità dell'attività, il tempo stimato, un valore di percezione della difficoltà della

task, la possibilità di scegliere il revisionatore dell'attività e tante altre opzioni, tutte queste attività possono essere poi raggruppate in milestones e a loro volta in progetti. Per ogni progetto è disponibile un diagramma Gantt autogenerato da Redmine stesso, questo per avere un'indicazione visiva dello stato del progetto.

Figura 2.7: logo Redmine

2.8 MySQL

Il database installato sul server aziendale è MySQL [12] (vedi Figura 2.8). Si tratta di un RDBMS[g](Relational Database Management System), ovvero un sistema di gestione di database basato sul modello relazionale prodotto da Oracle e distribuito gratuitamente.
Esso utilizza il linguaggio SQLG (Structured Query Language) per creare, visualizzare, reperire ed eliminare dati all'interno del database.
Per amministrare il database è stato utilizzato Webmin [19] (vedi Figura 2.9) ovvero un applicativo web scritto in Perl che consente di gestire database MySQL direttamente dal browser.

Figura 2.8: logo MySQL

Figura 2.9: logo Webmin

2.9 OAuth 2.0

OAuth2.0 [14] (vedi Figura 2.10) è un protocollo standard aperto che consente alle applicazioni di accedere alle risorse protette di un servizio per conto dell'utente. Questo protocollo più che una scelta è stato un requisito imposto dal cliente infatti ogni tipo di comunicazione con i loro endpoint doveva essere autenticata seguendo le specifiche del protocollo.
Per me OAuth2.0 è stata una tecnologia totalmente nuova di cui non conoscevo minimamente il funzionamento e per questo buona parte della formazione è stata dedicata al suo apprendimento.
È sicuramente una tecnologia interessate, e di larga diffusione, è infatti il protocollo standard per l'autorizzazione di applicazioni web, applicazioni desktop e smartphones

verso delle risorse protette.
Nella sezione 4.2 viene approfondito il flow del protocollo e la sua struttura.

Figura 2.10: logo OAuth2.0

Capitolo 3

Descrizione dello stage

In questo capitolo viene descritta la pianificazione e l'organizzazione dello stage

3.1 Obiettivi

Notazione

Si farà riferimento agli obiettivi secondo le seguenti notazioni:

* **O** per gli obiettivi obbligatori, vincolanti in quanto obbiettivo primario richiesto dal committente;
* **D** per gli obiettivi desiderabili, non vincolanti o strettamente necessari, ma dal riconoscibile valore aggiunto;
* **F** per gli obiettivi facoltativi, rappresentanti valore aggiunto ma non strettamente competitivo.

Obiettivi fissati

Gli obiettivi fissati sono i seguenti:

* Obbligatori
  - *O1*: Riuscire a riconoscere i requisiti principali e secondari da sviluppare nel progetto software estrapolandoli da comunicazioni mail e videochiamate;
  - *O2*: Riuscire a leggere e ad interpretare la documentazione di librerie e servizi esterni e stabilire la più efficace struttura dati per il tipo di comunicazione richiesta;
  - *O3*: Riuscire ad eseguire i principali test di cumunicazione e validità dei dati verso le API di Enel X [1];
  - *O4*: Riuscire ad eseguire il coretto parsing dei dati in formato JSON per trasmettere le informazioni secondo la struttura prefissata.
* Desiderabili
  - *D1*: Sviluppare la capacità di riuscire collaborare con il gruppo di lavoro;

  - *D2*: Sviluppare la capacità di analisi e risoluzione dei problemi.

* Facoltativi
  - *F1*: Acquisizione di autonomia nella realizzazione di moduli software;
  - *F2*: Ricerca autonoma di nuove tecnologie utili alle finalità del progetto.

3.2 Analisi preventiva dei rischi

Durante la fase di analisi iniziale sono stati individuati alcuni possibili rischi più probabili.
Si è quindi proceduto a elaborare delle possibili soluzioni per poterli affrontare.
1. Scarsa conoscenza del protocollo OAuth2.0 [14]
Descrizione: Le competenze riguardo al protocollo utilizzato per ottenere l'autorizzazione verso le API predisposte da Enel X [1] non sono sufficienti.
Soluzione: La prima parte del tirocinio consisterà nello studio e formazione di questo protocollo.

2. Forte dipendenza dal team di sviluppo di Enel X
Descrizione: Il completamento dell'applicativo è fortemente legato al team di sviluppo del cliente, il quale dovrà ultimare gli endpoint verso i quali inviare i dati.
Soluzione: Per tutta la durata dello stage, si mira a mantenere una comunicazione attiva e collaborativa nei confronti di Enel X.

3. Possibilità di testing limitate
Descrizione: É particolarmente rischioso testare l'applicativo, in fase di sviluppo, sui server di produzione collegati alla piattaforma di raccolta dati Tmacs.
É disponibile un server di test contenente un numero limitato di dati.
Soluzione: Porre particolare attenzione ai casi limite dell'aplicativo e assicurare, anche attraverso frequenti riunioni con il responsabile, la stabilità dello stesso.

3.3 Pianificazione

Lo stage è strutturato con obiettivi specifici distribuiti nel corso delle 8 settimane disponibili come riassunto alla Tabella 3.1.
Nelle prime settimane si affrontano maggiormente obiettivi come l'analisi del problema e lo studio delle tecnologie, per poi in una fase più avanzata, proseguire con la progettazione e lo sviluppo dell'applicativo stesso.

Attività	Dettaglio attività	Ore
Formazione	∗ Java EE [7] ∗ Resful JSON ∗ Customer API ∗ Protocollo OAuth2.0 [14] ∗ Strutture dati piattaforma TMacs [17]	40 ore
Progettazione	∗ Analisi fattibilità ∗ Sviluppo piano di progetto ∗ Progettazione strutture dati	40 ore
Sviluppo	∗ Predisposizione ambiente di sviluppo ∗ Realizzazione dell'applicativo	160 ore
Testing	∗ Test di comunicazione verso gli endpoint ∗ Test di scomposizione pacchetto ∗ Test di leggibilità dei dati	40 ore
Deploy	∗ Configurazione di un server linux di produzione ∗ Pubblicazione dell'applicativo	40 ore

Tabella 3.1: Attività preventivate

Cronoprogramma settimanale

Come anche mostrato in Figura 3.1, le attività sono state così suddivise:

- **Prima settimana - Formazione (40 ore)**
  - presentazione strumenti di lavoro per la condivisione del materiale di studio e per la gestione dell'avanzamento del percorso;

  - panoramica a livello commerciale del progetto e dei rapporti con Enel X [1];
  - studio della documentazione relativa alle API del cliente;
  - studio del protocollo di autenticazione OAuth2.0 [14];
  - studio e panoramica della piattaforma TMacs [17], in particolare il modulo Traffic, responsabile della raccolta dati provenienti dagli impianti semaforici a campo.

* **Seconda settimana - Progettazione (40 ore)**
  - riunioni congiunte con il responsabile per lo studio di fattibilità del progetto;
  - pianificazione delle fasi di progetto ed introduzione delle stesse su Redmine [16];
  - progettazione della struttura dati che poi dovrà essere serializzata ed inviata.

* **Terza settimana - Sviluppo (40 ore)**
  - configurazione dell'ambiente di sviluppo;
  - implementazione delle procedure di raccolta dati relative ai dati traffico veicolari.

* **Quarta settimana - Sviluppo (40 ore)**
  - implementazione della struttura dati di contenimento dei dati traffico veicolari e serializzazione della stessa.

* **Quinta settimana - Sviluppo (40 ore)**
  - implementazione delle procedure di raccolta dati relative agli stati semaforici.
  - implementazione della struttura dati di contenimento degli stati semaforici e serializzazione della stessa.

* **Sesta settimana - Sviluppo (40 ore)**
  - implementazione delle procedure per l'autenticazione con il protocollo OAuth2.0 [14];
  - implementazione delle procedure per l'invio dei dati serializzati, sia per i traffici veicolari che per gli stati.

* **Settima settimana - Test (40 ore)**
  - test di comunicazione con gli endpoint di Enel X;
  - test di validità e leggibilità dei dati;
  - test congiunti con Enel X per verificare la corretta ricezione e scomposizione del pacchetto ricevuto.

* **Ottava settimana - Deploy finale (40 ore)**
  - Pubblicazione dell'applicativo sui server di produzione.

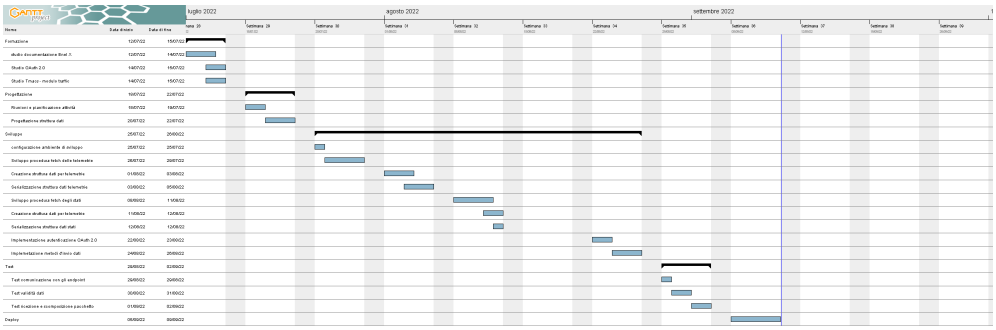

Figura 3.1: cronoprogramma settimanale

3.4 Organizzazione dello stage

Lavoro in sede

L'azienda permette ai propri dipendenti di lavorare esclusivamente in sede, ad eccezione di casistiche particolari. In ufficio i colleghi si sono sempre rivelati disponibili a condividere la loro esperienza lavorativa in un ambiente cordiale e socievole.

Meetings

Almeno una volta ogni due giorni circa, si sono svolte riunioni interne con il responsabile ed altri colleghi per verificare la correttezza del codice prodotto e per meglio coordinare le attività successive.

Interazione con Enel X

Enel X [1] è il committente del progetto, le comunicazioni con loro si sono svolte esclusivamente via email per tutta la durata dello stage.

Capitolo 4

Progettazione e codifica

In questo capitolo viene trattata la progettazione e la successiva realizzazione del prodotto software

4.1 Progettazione strutture dati

Prima di tutto è stato necessario realizzare, in collaborazione con Enel X [1], un mock up delle strutture dati che poi sarebbero state serializzate nel formato JSON.
I mock up realizzati si concentravano sul trovare un compromesso tra la semplicità di realizzazione e la completezza delle informazioni ricevute.

4.1.1 Telemetrie

Premesse

Le telemetrie rappresentano i dati provenienti da ogni singolo sensore installato nell'intersezione semaforica (come ad esempio spire e radar), esse vengono registrate dal regolatore semaforico e scritte in un database MySQL [12].
Le telemetrie sono molto utili nell'ambito dell'ingegneria del traffico in quanto riescono a fotografare lo stato dell'intersezione semaforica ad ogni istante, fornendo informazioni sull'intesità dei flussi veicolari.

Dati richiesti da EnelX

Enel X [1] ha espresso la necessità di poter ottenere da ogni singolo sensore di ogni regolatore semaforico in Italia, ad aggregati temporali di 5 minuti, i seguenti dati (che loro chiamano *misurazioni*):

* numero di veicoli all'ora;
* velocità media;
* lunghezza media;
* numero di moto all'ora;

* numero di macchine all'ora;
* numero di van all'ora;
* numero di mezzi pesanti all'ora;
* numero di bus all'ora;
* numero di veicoli non classificati all'ora.

Tutti questi dati vengono registrati, per la maggior parte, attraveso delle spire a conduzione magnetica poste in prossimità dell'intersezione semaforica. Ogni spira è in grado di misurare la lunghezza del veicolo che la attraversa e questo ci permette di poter classificare i veicoli.
Alle telemetrie inoltre, sono stati aggiunti dati di supporto, come il timestamp di quando viene generato il JSON, il paese da cui sono stati prelevati i dati, la timezone e gli identificativi (detti UUID[g], Universally unique identifier) del sensore che ha registrato i dati.

Struttura dati

Nella Figura 4.1 viene riportato l'UML[g](Unified Modeling Language) delle classi elencate di seguito, l'UML è un linguaggio di modellazione e di specifica basato sul paradigma orientato agli oggetti.
I dati precedentemente elencati sono stati organizzati nelle seguenti classi Java [7]:

* it.tecsen.enelxapi.v2.telemetries.model.**ParsableBody**;
* it.tecsen.enelxapi.v2.telemetries.model.**ParsableDevice**;
* it.tecsen.enelxapi.v2.telemetries.model.**ParsableMeasure**;
* it.tecsen.enelxapi.v2.telemetries.model.**ParsableTelemetry**.

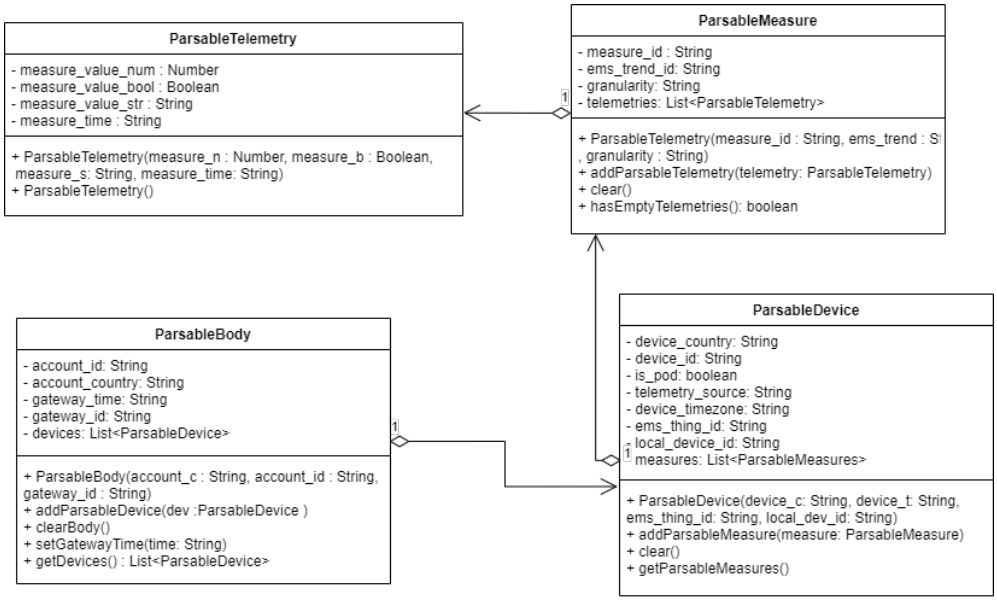

Figura 4.1: UML del package it.tecsen.enelxapi.v2.telemetries.model

Queste classi, attraverso la libreria di Google **Gson** [6], sono state serializzate in un file JSON così composto:


```
{
  "account_id": "##########################",
  "account_country": "IT",
  "gateway_time": "2022-08-29T19:15:00.000",
  "devices": [
    {
      "device_country": "IT",
      "device_id": "0000-0079-0032-001E
          -0031-0001-0034-0001_0000-0008-FFFF-FFFF_IT_tmacs",
      "is_pod": false,
      "telemetry_source": "tmacs",
      "device_timezone": "Europe/Rome",
      "ems_thing_id": "0000-0079-0032-001E
          -0031-0001-0034-0001_0000-0008-FFFF-FFFF",
      "measures": [
        {
          "measure_id": "37200010",
          "ems_trend_id": "37200010",
          "granularity": "5m",
          "telemetries": [
            {
              "measure_value_str": "",
              "measure_time": ""
            }
          ]
        },
        {
          "measure_id": "37200000",
          "ems_trend_id": "37200000",
          "granularity": "5m",
          "telemetries": [
            {
              "measure_value_num": 72,
              "measure_time": "2022-08-29T19:15:00.000"
            },
            {
                "measure_value_num": 24,
                "measure_time": "2022-08-29T19:20:00.000"
            },
            {
                "measure_value_num": 12,
                "measure_time": "2022-08-29T19:25:00.000"
            },
            {
                "measure_value_num": 53,
                "measure_time": "2022-08-29T19:30:00.000"
            }
          ]
        },
        {
          "measure_id": "37200011",
          "ems_trend_id": "37200011",
```

```
      "granularity": "5m",
      "telemetries": [
        {
          "measure_value_str": "",
          "measure_time": ""
        }
      ]
    },
    {
      "measure_id": "37200004",
      "ems_trend_id": "37200004",
      "granularity": "5m",
      "telemetries": [
        {
          "measure_value_str": "",
          "measure_time": ""
        }
      ]
    },
    {
      "measure_id": "37200005",
      "ems_trend_id": "37200005",
      "granularity": "5m",
      "telemetries": [
        {
          "measure_value_str": "",
          "measure_time": ""
        }
      ]
    },
    {
      "measure_id": "37200006",
      "ems_trend_id": "37200006",
      "granularity": "5m",
      "telemetries": [
        {
          "measure_value_str": "",
          "measure_time": ""
        }
      ]
    },
    {
      "measure_id": "37200007",
      "ems_trend_id": "37200007",
      "granularity": "5m",
      "telemetries": [
        {
          "measure_value_str": "",
          "measure_time": ""
        }
      ]
    },
    {
      "measure_id": "37200008",
```

```
          "ems_trend_id": "37200008",
          "granularity": "5m",
          "telemetries": [
            {
              "measure_value_str": "",
              "measure_time": ""
            }
          ]
        },
        {
          "measure_id": "37200009",
          "ems_trend_id": "37200009",
          "granularity": "5m",
          "telemetries": [
            {
              "measure_value_str": "",
              "measure_time": ""
            }
          ]
        }
      ]
    }
  ]
```


4.1.2 Stati

Premesse

Gli stati semaforici rappresentano i dati realtime, sull'attuale funzionamento e codizione dell'impianto semaforico.
Ogni regolatore semaforico, attraverso il firmware[g] che lo controlla, registra costantemente dati come il suo funzionamento, il piano semaforico e le diagnostiche.
In particolare, le diagnostiche rappresentanto un log nel quale vengono raggruppate le informazioni su eventuali guasti software o hardware del regolatore, o di tutti gli apparati semaforici esterni ad esso collegati (e.g lanterne semaforiche, radar, spire).

Dati richiesti da Enel X

Inizialmente Enel X [1] ha espresso la necessità di ottenere i dati di:

* **Funzionamento**: ovvero l'attuale stato operativo dell'intersezione semaforica, alcuni esempi sono:

  1. Spento;
  2. Funzionamento a tutto rosso;
  3. Funzionameno a lampeggio;
  4. Funzionamento a tempi fissi;
  5. Funzionamento in stato pilotato;
  6. Funzionamento in stato sincronizzato;

* **Piano**: ovvero una lista di funzionamenti schedulati che identifica il comportamento dell'intersezione semforica in dato un giorno, o ora del giorno. I piani semaforici vengono impostati da un tecnico, e rimangono tali a meno di interventi speciali. Generalmente un regolatore riesce a gestire 16 piani semaforici distinti;

* **Ptpriorities**: assume due valori, abititato o disabilitato, ed indica il preferenziamento verso i mezzi pubblici come ad esempio ambulanze o auto della polizia;

* **Diagnostica**: ovvero i log registrati dal regolatore, che tengono traccia di tutti i malfunzionamenti hardware e software. A seconda dell'errore, il rgolatore genera log a più alta o più bassa priorità. Un esempio è un carico di corrente inferiore proveniente dalle lanterne che si traduce in *lampada guasta*.

A seguito però di alcune sue valutazioni interne, il cliente ha deciso di escludere i dati di diagnostica rispetto a quanto stabilito all'inizio del progetto.

Struttura dati

I dati precedentemente elencati sono stati organizzati nelle seguenti classi Java [7] (vedi Figura 4.2):

* it.tecsen.enelxapi.v2.states.model.**ParsableBody**;

* it.tecsen.enelxapi.v2.states.model.**MessageList**;

* it.tecsen.enelxapi.v2.states.model.**Message**;

* it.tecsen.enelxapi.v2.states.model.**ParsableAnswerState**;

* it.tecsen.enelxapi.v2.states.model.**ParsableTrafficController**.

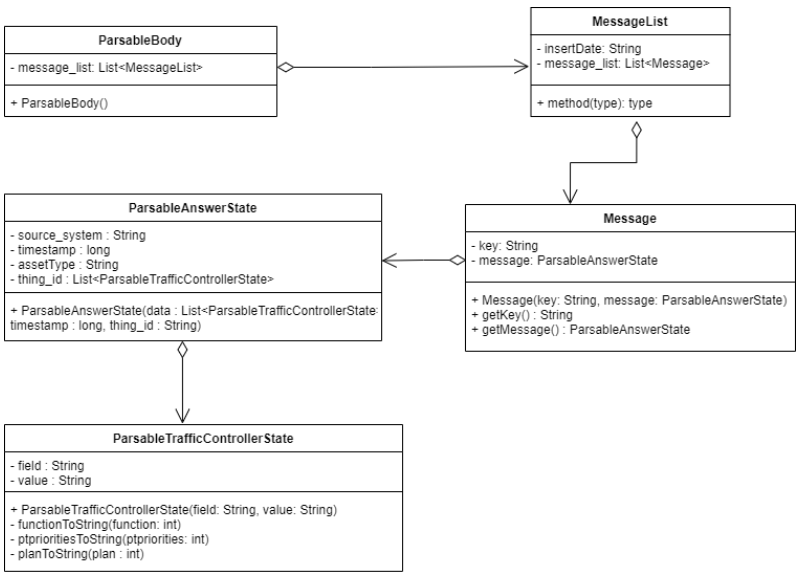

Figura 4.2: UML del package it.tecsen.enelxapi.v2.states.model

Queste classi, attraverso la libreria di Google **Gson** [6], sono state serializzate in un file JSON così composto:

```
{
  "message_list": [
    {
      "message": {
        "source_system": "tmacs",
        "timestamp": 1666979712967,
        "assetType": "Semaphoric Gateway",
        "thing_id": "0000-0079-0032-001E-0031-0001-FFFF-FFFF",
        "data": [
          {
            "field": "function",
            "value": "Funzionamento in stato attuato"
          },
          {
            "field": "plan",
            "value": "Piano semaforico 0"
          },
          {
            "field": "ptpriorities",
            "value": "Preferenziamento mezzo pubblico Abilitato"
          }
        ]
      }
    }
  ]
}
```

4.1.3 Configurazione

Premesse

Nella fase di progettazione è stata individuato il bisogno di rendere l'applicativo configurabile.
Le motivazioni principali che abbiamo individuato sono state: poter indicare da quali database e da quali impianti prelevare i dati; avere un file che temga traccia delle informazioni necessarie per connettersi agli endpoint.

Dati richiesti da Tecsen

Analizzando il file di configurazione nel quale si registrano gli impiani semaforici con cui interagire, troviamo informazioni come ad esempio:

* il codice del database da cui prelevare i dati;
* il codice identificativo del sensore;
* il codice identificativo del tipo di sensore;

Nella parte riguardante la configurazione per la connessione verso gli endpoint di EnelX, abbiamo inserito informazioni come:

* l'host verso il quale connettersi

* il client id e la client secret che permettono l'autenticazione verso l'endpoint di autoirizzazione Oauth2.0 [14];

Questi dati vengono impostati ed aggiornati da un tecnico interno a Tecsen [17] nel caso si presentasse il bisogno.

Struttura dati

Trattandosi di una situazione invertita rispetto ai due casi precedenti, abbiamo prima definito la struttura di un file JSON che poi viene letto e deserializzato in specifiche classi Java [7].
Il file JSON è così strutturato:

```
{
    "request_environtment": {
        "id": "##################",
        "values": [
            {
                "key": "host",
                "value": "###################",
            },
            {
                "key": "client_id",
                "value": "###################",
            },
            {
                "key": "client_secret",
                "value": "###################",
            }
        ]
    },
    "archives": [
        {
            "db_code": 2,
            "devices": [
                {
                    "device_id": 1,
                    "device_type": 20,
                    "component_id": 1,
                    "component_type": 161,
                    "subcomponent_id": -1,
                    "subcomponent_type": -1
                }
            ]
        }
    ]
}
```

Le classi Java [7] nelle quali il file viene deserializzato sono le seguenti (vedi Figura 4.3):

* it.tecsen.enelxapi.v2.conf.**Configuration**;
* it.tecsen.enelxapi.v2.conf.**Database**;
* it.tecsen.enelxapi.v2.conf.**RequestEnvironment**;

* it.tecsen.enelxapi.v2.conf.**NoProdProperties**;
* it.tecsen.enelxapi.v2.conf.**SemaphoreRegulator**.

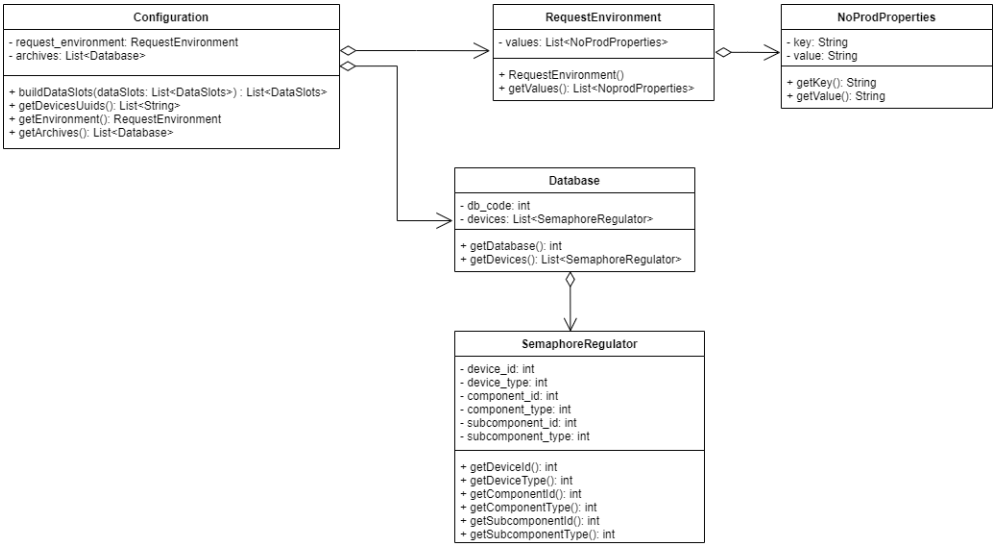

Figura 4.3: UML del package it.tecsen.enelxapi.v2.conf

4.2 Protocollo OAuth 2.0

Una parte importante affrontata in fase di progettazione è stata quella di apprendere il funzionamento del framework di autenticazione Oauth2.0 [14].
Il framework viene utilizzato da Enel X [1] per autenticare e autorizzare le richieste HTTP in arrivo dai server di produzione Tecsen.

4.2.1 Formazione

Durante le prime settimane, oltre alla progettazione del software, si è dato spazio anche alla formazione riguardo il funzionamento della tecnlogia in questione.
La formazione è stata strettamente necessaria, non solo per quanto riguarda l'utilizzo effettivo del protocollo nell'applicativo, ma anche per interesse dell'azienda stessa che ne ha intravisto le potenzialità per applicazioni future.
La risorsa principale che è stata utilizzata per apprendere il funzionamento del protocollo è la documentazione ufficiale di OAuth2.0 [14].

Protocol Flow

Nella Figura 4.4 vengono illustrati gli steps richiesti dal protocollo per autorizzare un client all'utilizzo di una specifica risorsa. Gli steps sono i seguenti:

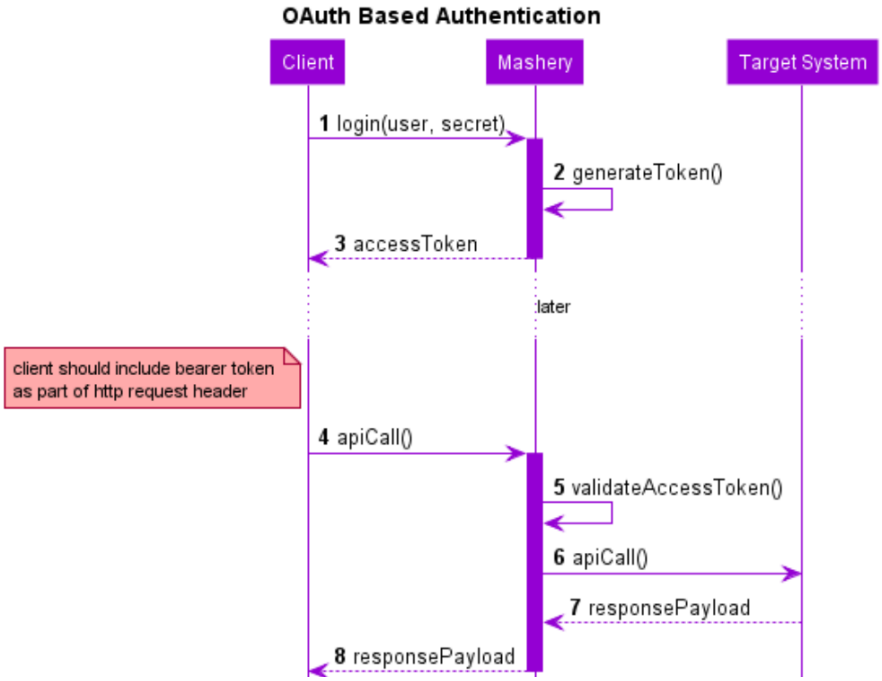

Figura 4.4: Oauth2.0 protocol flow

1. il client invia una richiesta di login verso il layer di autenticazione, in questo caso fornito da un servizio di terze parti chiamato Mashery [10]. I parametri richiesti per autenticarsi correttamente sono **user** e **secret**.

User e secret in questo caso sono forniti dalla stessa Enel X [1], solo ed esclusivamente agli indirizzi ip dei server Tecsen [17], e sono memorizzati nel file di configuazione approfondito al Capitolo 4.1.3;

2. il servizio Mashery [10], una volta validati user e secret, produce un **access-token**;
3. a questo punto il client richiede la risorsa protetta al target system, autenticandosi attraverso l'access-token appena ottenuto;
4. in fine il servizio di autenticazione valida l'access-token, e se valido, inoltra la richiesta al target system che la esegue e restituisce un payload di risposta.

OAuth2.0 [14] quindi consente a diversi tipi di applicazioni (client) di verificare l'identità dell'utente attraverso l'autenticazione per mezzo di un server di autorizzazione e oltre a ciò di ottenere ulteriori informazioni di base sull'utente.
In cambio, il protocollo viene completato con tutte le funzioni necessarie per il login e il Single sign-on.

Access-token

Gli access-token sono le credenziali utilizzate per accedere a risorse protette. Un access-token di fatto è una stringa che rappresenta una specifica autorizzazione rilasciata al client.
Ogni access-token in se stesso specifica un determinato scopo per il quale è stato richiesto e la sua durata (di solito un'ora).
Esistono access token di differenti formati, struttura e metodi di utilizzo in base ai requisiti dei server di autenticazione. Un esempio sono i refresh-token che a differenza di quelli descritti precedentemente hanno una durata più prolungata, e una volta scaduti, possono essere scambiati con nuovi access-token; proprio per le proprietà appena descritte questi token devono essere memorizzati in maniera sicura dai client che ne fanno utilizzo.

4.3 Struttura software

L'applicativo è stato realizzato in Java [7] seguendo una specifica struttura dei package. Ogni package viene figurato come una funzionalità principale che l'applicativo deve soddisfare, e per ogni funzionalità che lo richiede la si scompone in sottopackage aggiuntivi.
I package come *telemetries* e *states*, ad esempio, si suddividono a loro volta in: **logic**, contenente la logica elaborativa di tutto il modulo, e **model**, contenente la struttura dati per la memorizzazione delle informazioni.

Quasi ogni package principale si traduce poi in uno specifico thread in costante esecuzione che viene avviato e distrutto da una classe, chiamata **EnelXManagerBean.java**, la quale applica il design pattern creazionale *Singleton*.
Nella Figura 4.5 viene illustrato il diagramma dei package e più in generale l'architettura dell'applicativo, per facilità rappresentativa, il package *commons* non è stato interconnesso con gli altri in quanto tutti ne fanno utilizzo e connetterlo porterebbe ad una rappresentazione confusa.

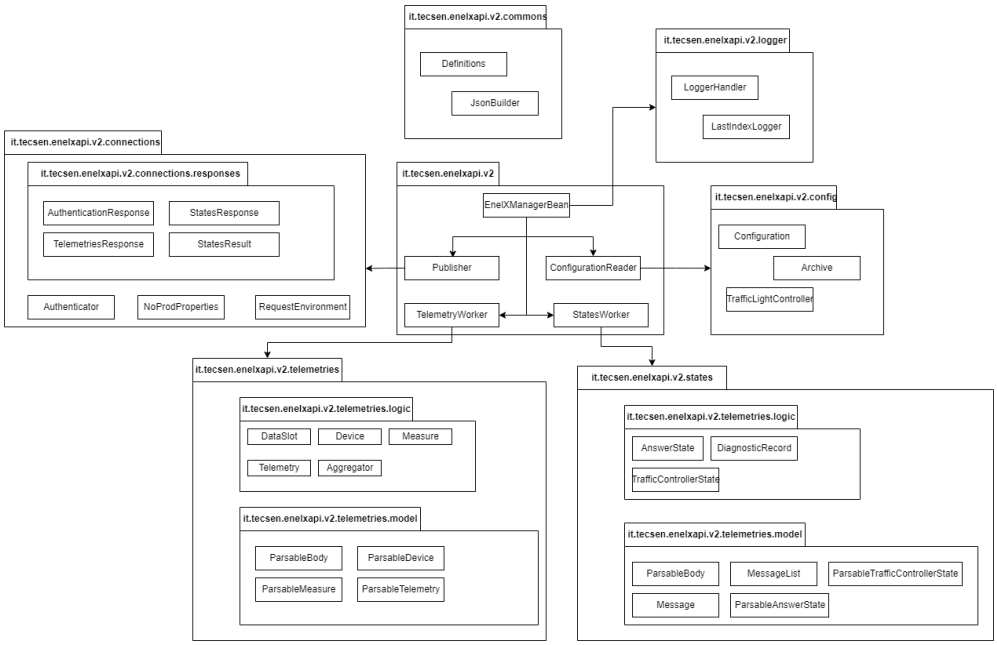

Figura 4.5: Diagramma dei package dell'applicativo

L'applicativo inoltre segue in modo molto preciso le convenzioni sintattiche e semantiche condivise da Google e chiamate Google Style Guides [4], questo per mantenere un grado di leggibilità e manutenibilità elevato.

4.3.1 Implementazione

Di seguito vengono descritti a livello implementativo i package che compongono l'applicativo soffermandosi sulle classi principali.

it.tecsen.enelxapi.v2

È il package principale che contiene la classe *EnelXManagerBean.java* ovvero il "main" dell'applicativo, definita secondo il design pattern **Singleton** in quanto deve esistere in una ed una sola istanza e deve essere inizializzata esclusivamente all'avvio dell'applicativo.
Questa classe ha lo scopo di inizializzare, istanziare e distruggere tutti i thread principali che compongono l'applicativo.
Il package contiene inoltre le classi:

* *TelemetryWorker.java*;
* *StatesWorker.java*;
* *ConfigurationReader.java*;
* *Publisher.*

Queste classi implementano tutte l'interfaccia java.lang.**Runnable** che le rende task eseguibili in un Thread.
ConfigurationReader.java si occupa di leggere, circa una volta ogni ora, il file di configurazione dell'applicativo in modo tale che automaticamente carichi i dati, come ad esempio nuovi sensori da interrogare.

TelemetryWorker.java è la classe che costantemente, circa ogni 5 minuti, interroga il database allo scopo di ottenere nuovi dati provenienti dai sensori a campo. Richiama i metodi definiti nelle classi presenti al package 4.3.1 per l'elaborazione e la conseguente produzione del payload in formato JSON.

StatesWorker.java in maniera molto simile alla classe precedente, interroga il database ogni quindici secondi circa per ottenere i dati più aggiornati, confronta i dati appena ottenuti con quelli inviati la volta precedente e se trova delle differenze invia una nuova richiesta verso l'endpoint.

Publisher.java è la classe che si occupa di richiedere un nuovo autentication token dopo che quello precedente è scaduto.
Predispone inoltre il metodo per generare ed eseguire le richieste cURL verso i rispettivi endpoint di telemetrie o di stato.

it.tecsen.enelxapi.v2.logger

In questo package è contenuta la classe *LoggerHandler.java* che definisce il comportamento di un logger, istanziato staticamente in tutte le altre classi, utilizzato per scrivere log su file *.txt, in una specifica cartella del server.
È inoltre presente la classe *LastIndexLogger.java* che predispone dei metodi statici per la lettura e la scrittura di un file, *lastindexes.properties*, contenente per ogni sensore analizzato l'id dell'ultimo record ottenuto dal database ed inviato all'endpoint.

La motivazione che ci ha spinto a creare la classe precedentemente spiegata è che in caso di undeploy dell'applicativo dal server, una volta ripartito, non debba ricominciare ad inviare tutti i dati nuovamente ma semplicemente possa ripartire dagli ultimi salvati.

it.tecsen.enelxapi.v2.telemetries

Questo package si scompone in due sottopackage.

it.tecsen.enelxapi.v2.telemetries.logic

Contiene le classi:

* *Aggregator.java*;
* *DataSlot.java*;
* *Device.java*;
* *Measure.java*;
* *Telemetry.java*;

Queste classi vanno a definire la logica relativa ai dati di telemetria.
In particolare in questo package analizzando i dati ottenuti dalla classe *TelemetryWorker.java* vengono generati i device (sensori), che a loro volta contengono una lista di nove misurazioni, le quali contengono una lista di telemetrie suddivise a slot temporali di 5 minuti. Ogni misurazione è identificata nel file JSON con un corrispettivo codice, concordato all'inizio del progetto con il cliente Enel X [1].

DataSlot.java rappresenta il nuovo slot di dati provenienti da *TelemetryWorker.java* e al suo interno si occupa di smistare ogni dato nel corrispondente oggetto di tipo *Device* confrontando gli identificativi.
Ogni dato salvato sul database contiene le informazioni che ne identificano la sua provenienza quali ad esempio:

* **device_id**: codice identificativo dell'impianto;
* **device_type**: codice identificativo del tipo di impianto;
* **component_id**: codice identificativo dello specifico componente (e.g. spire, radar) appartentente all'impianto;
* **component_type**: codice identificativo del tipo di componente;
* **subcomponent_id**: codice identificativo del modulo di un componente (-1 se sconosciuto);
* **subcomponent_type**: codice identificativo del tipo di sottocomponente (-1 se sconosciuto);
* **db_code**: codice identificativo del database;
* **measurement_point**: indice della corsia stradale.

Attraverso queste informazione è possibile generare un codice identificativo univico, chiamato UUID, associato ad ogni oggetto di tipo *Device.java*.

```
0000-0079-0032-001E-0031-0001-FFFF-FFFF
```

Attraverso la classe *Aggregator.java* viene prodotto per ogni solt temporale, quindi per ogni telemetria, un aggregato.

Esistono regolatori che sono in grado, già attraverso il loro firmware di generare aggregati veicolari, mentre altri che registrano e scrivono sul database veicolo per veicolo.

Nel secondo caso, *Aggregator.java* semplicemente raggruppa tutti i dati provenienti da uno stesso sensore e in uno stesso slot temporale e ne calcola informazioni come: velocità media, numero di veicoli all'ora, numero di camion all'ora, ecc.

Le classi presenti in questo package si occupano anche di istanziare e popolare tutte le classi che compongono la struttura nella quale verranno memorizzate le informazioni elaborate.

it.tecsen.enelxapi.v2.telemetries.model

In questo package sono contenute tutte le classi che compongono la struttura dati nella quale vengono memmorizzati i dati elaborati.

Ogni classe dispone un costruttore e dei metodi getter, tutti i campi di classe sono dichiarati final, e attraverso la libreria **Gson** [6] vengono serializzati in un oggetto di tipo JSON, come approfondito al Capitolo 4.1.1.

it.tecsen.enelxapi.v2.states

Questo package si scompone in due sottopackage.

it.tecsen.enelxapi.v2.states.logic

Contiene le classi:

* *AnswerState.java*;
* *TrafficControllerState.java*

AnswerState.java è una classe contenete la lista degli utlimi stati, ottentuti dall'interrogazione verso il database, gli stati sono rappresentati dalla classe *TrafficControllerState.java*.
Quest'ultima classe predispone una serie di metodi utili a *StatesWorker.java*, ad esempio esegue l'override del metodo di default **equals** per permettere la comparazione degli oggetti di tipo TrafficControllerState utilizzando il campo **uuid**.

it.tecsen.enelxapi.v2.states.model

Contiene le classi:

* *ParsableBody.java*;
* *MessageList.java*;
* *Message.java*;
* *ParsableAnswerState.java*;

* *ParsableTrafficController.java.*

In questo package sono contenute tutte le classi che compongono la struttura dati nella quale vengono memmorizzati le informazioni elaborate, come approfondito al Capitolo 4.1.2.

it.tecsen.enelxapi.v2.connections

Questo package contiene le seguenti classi:

* *Authenticator.java*;
* *RequestEnvironment.java*;
* *NoProdProperties.java.*

Più precisamente *Authenticator.java* è la classe che, attraverso la composizione, utilizza le proprietà definite nell'oggetto RequestEnvironment per permettere l'autenticazione verso l'endpoint di EnelX.
L'endpoint di Enel X [1], a meno di errori nella richiesta o nella comunicazione, restituisce una risposta contentente l'autentication token il quale permetterà alle successive richieste verso l'endpoint di essere riconosciute come provenienti da una fonte autenticata.

Più precisamente la classe *Authenticator.java* descritta in precedenza si occupa di leggere i dati deserializzati dal file di configurazione nella classe *RequestEnvironment.java* e di comporre ed eseguire la richiesta cURL verso il servizio di autenticazione.
La richiesta cURL è così composta:

```
curl -X POST https://host/tmacs/oauth2/token
-H "Content-type application/x-www-form-urlencoded"
-H "Authorization basic base_64_credentials"
-d "grant_type=client_credentials"
```

La classe si occupa anche di, una volta ottenuta la risposta, deserializzarla in un oggetto di classe *AuthenticationResponse.java* che sarà poi utilizzato da *Publisher.java* per effettuare richieste autorizzate verso le risorse protette.

it.tecsen.enelxapi.v2.connections.responses

Questo package contiene le seguenti classi:

* *AuthenticationResponse.java*;
* *StatesResponse.java*;
* *StatesResult.java*;
* *TelemetriesResponse.java*;

Quando si invia una richiesta di autenticazione o di invio dati verso il server di Enel X [1], quest'utltimo risponde con una stringa in formato JSON contentente diverse informazioni.
Alcune tra queste sono ad esempio: il codice della risposta, il messaggio, il codice di autenticazione, la durata del codice di autenticazione ecc.

Tutte queste informazioni vengono deserializzate nelle classi precedentemente elencate, in modo da poterle utilizzare nell'applicativo, per ad esempio verificare se una richiesta è andata a buon fine o meno.

it.tecsen.enelxapi.v2.states.commons

Contiene le classi:

* *Definitions.java*
* *JsonBuilder.java*

Definition.java è un banale contenitore di costanti utili al resto delle classi presenti nell'applicativo. Troviamo ad esempio i codici identificativi delle diverse misurazioni, o ad esempio i codici e le corrispondenti descrizioni verbali dei funzionamenti semaforici.

JsonBuilder.java è invece la classe che si occupa di istanziare l'oggetto **Gson**[6] e di fornire i metodi per la serializzazione e la deserializzazione sia dei payloads che delle risposte.

Capitolo 5

Verifica e validazione

In questo capitolo vengono descritte le attività di validazione e verifica

5.1 Premesse

La verifica e la validazione del software servono per accertare che tale software rispecchi e rispetti i requisiti analizzati durante la fase di analisi. Nei casi migliori queste attività sono molto complesse e seguono delle specifiche regole, o buone pratiche, al fine di ottenere il maggior grado di copertura.
Con grado di copertura si intende la percentuale di codice che è stata verificata e validata attraverso i test che sono stati predisposti.
Test che garantiscono un altro grado di copertura del codice assicurano anche un software con una minore densità di bug, più leggibile e più facilmente manutenibile.

Durante l'esperienza di stage questa metodologia di verifica e validazione non è stata applicata bensì, a causa di una ridotta quantità di tempo, sono stati eseguiti test molto più semplificati.

5.1.1 Test eseguiti

La verifica e validazione dell'aplicativo si è svolta nel corso della penultima settimana di stage, in particolare sono stati effettuati i seguenti test:

Test di comunicazione

L'applicativo, una volta completato, è stato caricato su un server di test appositamente predisposto allo scopo di creare un ambiente sicuro su cui eseguire il software, e che permettesse di effettuare tutte le attività di validazione e verifica previste senza compromettere il normale funzionamento di un server di produzione ufficiale.
L'applicativo è stato poi eseguito e come prima attività di verifica si è controllato lo stato delle connessioni verso gli endpoint di Enel X[1], più precisamente, una volta accertato il funzionamento corretto in situazioni ordinarie, sono stati simulati diversi casi limite per controllare la risposta dell'applicativo.
Alcuni tra questi casi limite sono stati ad esempio:

* *undeploy dell'applicativo:* non è raro infatti che l'applicativo, per manutenzione o a causa di un errore, venga fermato o addirittura rimosso dal server.

In questi casi il software prima di essere stoppato salva l'ultimo slot di dati che ha inviato agli endpoint, sia per le telemetrie che per gli stati, e lo scrive su un file chiamato *lastindexes.properties*. In questo modo, una volta ripartito, secondo i requisiti del cliente il software deve essere in grado di riaprire le connessioni, e di riprendere la sua esecuzione in base a quanto salvato nel file precedentemente menzionato;

* *endpoint non raggiungibile:* non sono tanto meno rari i problemi di comunicazione, infatti il sofware, se per qualsiasi motivo dovesse rilevare come irraggiungibile l'endpoint, è in grado di memorizzare internamente gli ultimi 5 slot (come richiesto da Enel X[1]) di dati e di ritardare l'invio sino a quando la connessione non riprende.

Test di scomposizione del pacchetto

I test di scomposizione del frame comprendo tutte quelle attività relative alla validità dei dati trasmessi. È tutto nell'interesse di Tecsen [17] che i dati inviati siano corretti non solo nella forma ma anche nel contenuto, per questo motivo, sia per telemetrie che per gli stati, sono stati analizzati i pacchetti inviati dall'applicativo.
Nel caso delle telemetrie ad esempio sono state analizzate tutte le misurazioni disponibili ovvero:

* numero di veicoli all'ora;
* velocità media;
* lunghezza media;
* numero di moto all'ora;
* numero di macchine all'ora;
* numero di van all'ora;
* numero di mezzi pesanti all'ora;
* numero di bus all'ora;
* numero di veicoli non classificati all'ora.

Per ciascuna di queste misurazioni, prelevando i dati attraverso l'applicativo di Tecsen [17] già funzionante e collaudato chiamato TMacs [17], sono stati confrontati i dati degli stessi impianti semaforici allo scopo di assicurarsi la loro correttezza. Nel caso degli stati semaforici è stata eseguita la medesima procedura, scomponendo il pacchetto inviato e analizzando ogni signola informazione trasmessa.
Questa procedura è stata ripetuta per un numero limitato ma sufficiente di volte garantendo così, almeno nella maggioranza dei casi, dati corretti.

Test di leggibilità

I test di leggibilità sono test molto semplici ma comunque importanti, voglio assicurare il fatto che il contenuto dei pacchetti inviati, non solo sia corretto, ma anche comprensibile al cliente. I pacchetti sono stati revisionati e valutati insieme al cliente sia per quanto riguarda la struttura del file e sia per quanto riguarda la leggibilità dei dati, in

particolare quelli con una traduzione testuale come ad esempio il dato *ptpriorities* del pacchetto di stati.
È stata controllata inoltre la validità dei timestamp, ovvero una sequenza di caratteri che rappresentano una data e/o un orario, soprattutto nel loro formato. Come da specifiche imposte dal cliente la data e l'ora devono essere rappresentate in formato UTC (Coordinated Universal Time) esteso, ovvero il classico formato UTC ma con una precisione aumentata sino al millisecondo.

Capitolo 6

Conclusioni

In questo capitolo vengono tratte le conclusioni sull'esperienza di stage.

6.1 Consuntivo delle attività

In questa sezione vengono evidenziate le differenze tra il programma inizialmente preventivato e ciò che effettivamente è stato fatto.

Attività	**Ore a preventivo**	**Ore a consuntivo**
Formazione	40 ore	40 ore
Progettazione	40 ore	42 ore
Sviluppo	160 ore	163 ore
Testing	40 ore	35 ore
Deploy	40 ore	40 ore
Totale	320 ore	320 ore

Tabella 6.1: Consuntivo delle attività

Come ben evidenziato nella Tabella riassuntiva 6.1, il primo leggero scostamento rispetto alla pianificazione si è verificato nella fase di progettazione in quanto il cliente ancora non aveva un'idea chiara di quali dati gli fossero necessari.
In particolare, per quanto riguarda gli stati semaforici, l'idea inizialmente concordata da entrambe le parti, era quella di includere anche i dati di diagnostica dei regolatori ma a progettazione già conclusa ci è stato richiesto di rimodellare la struttura escludendo quest'ultimi.

Anche nella fase di sviluppo ci sono stati dei rallentamenti, se pur non così significativi, causati da un ritardo nella predisposizione degli endpoint da parte di EnelX, soprattutto quello relativo ai dati di telemetria, e di conseguenza lo sviluppo del package *connections* (vedi Sezione 4.3.1) è stato posticipato.
La fase di testing invece ha avuto un impatto minore sulle tempistiche rispetto a quanto

pianificato visto che l'applicativo ha subito risposto bene alla maggior parte dei casi limite che ci eravamo prefissati di gestire.

6.2 Gestione delle criticità

In merito alle criticità riportate alla Sezione 1.4, siamo riusciti ad adattare la pianificazione delle attività.
La prima criticità menzionata non è stata propriamente risolta in quanto si trattavano comunque di richieste da parte del cliente stesso e quindi, anche se non previste inizialmente, le abbiamo implementate ugualmente. Abbiamo però sempre cercato, attraverso una sana collaborazione, di trovare ad ogni richiesta fuori specifica un compromesso che potesse giovare ad entrambe le parti.

La seconda criticità riguardante la limitata possibilità di testing dell'applicativo è stata risolta configurando un server di test, con un database dedicato al progetto nel quale sono stati caricati i dati storicizzati dei due mesi antecedenti, provenienti da circa 50 intersezioni semaforiche. I dati sono stati prelevati dai server di produzione utilizzando l'applicativo che l'azienda ha sviluppato per il monitoraggio e il pilotaggio remoto delle intersezioni semaforiche.

Riguardo all'ultima problematica, essendo dovuta a questioni interne ad Enel X[1], abbiamo leggermente modificato la pianificazione temporale iniziale dando priorità agli altri package dell'applicativo che non dipendevano dagli endpoint. Abbiamo inoltre fornito il massimo supporto nei confronti dell'azienda committente nel cercare, attraverso dei test comunicativi, di riportare eventuali anomalie sugli endpoint.

6.3 Raggiungimento degli obiettivi

Il raggiungimento degli obiettivi (vedi Tabella 6.2), indicati alla Sezione 3.1, è da ritenere soddisfacente.
Lo sviluppo di un applicativo center to center in Java mi ha permesso di ottenere maggiori competenze per quanto riguarda l'interazione con i database e la serializzazione di oggetti in JSON.
La fase di pianificazione mi ha permesso di cimentarmi con vari problemi tra i quali la gestione di imprevisti e la suddivisione temporale di un progetto.
La gestione della comunicazione con i clienti e del rapporto collaborativo con essi sono stati ritenuti più che soddisfacenti dall'azienda.

Codice	**Descrizione**	**Stato**
O1	Riuscire a riconoscere i requisiti principali e secondari da sviluppare nel progetto software estrapolandoli da comunicazioni mail e videochiamate	Soddisfatto

O2	Riuscire a leggere e ad interpretare la documentazione di librerie e servizi esterni e stabilire la più efficace struttura dati per il tipo di comunicazione richiesta	Soddisfatto
O3	Riuscire ad eseguire i principali test di cumunicazione e validità dei dati verso le API di Enel X[1]	Soddisfatto
O4	Riuscire ad eseguire il coretto parsing dei dati in formato JSON per trasmettere le informazioni secondo la struttura prefissata.	Soddisfatto
D1	Sviluppare la capacità di collaborare con il gruppo di lavoro	Soddisfatto
D2	Sviluppare la capacità di analisi e risoluzione dei problemi	Soddisfatto
F1	Acquisizione di autonomia nella realizzazione di moduli software	Soddisfatto
F2	Ricerca autonoma di nuove tecnologie utili alle finalità del progetto	Soddisfatto

Tabella 6.2: Tabella degli obiettivi obbligatori, desiderabili e facoltativi soddisfatti

6.4 Conoscenze personali pregresse

Nella Tabella 6.3 vengono riassunte le conoscenze possedute, a monte dello stage, riguardo a quelle che poi sono state le tecnologie utilizzate.

Tecnologia	**Livello pre-stage**	**Livello post-stage**
NetBeans [13]	Buono	Buono
Git [2]	Buono	Buono
GitAhead [3]	Nullo	Scarso
Payara [15]	Scarso	Buono
Java [7]	Buono	Buono
Gson [6]	Scarso	Buono
Redmine [16]	Nullo	Scarso
MySQL [12]	Buono	Buono
Webmin [19]	Scarso	Scarso
OAuth2.0 [14]	Nullo	Buono

Tabella 6.3: Conoscenza personale delle tecnologie utilizzate nel progetto

6.5 Conoscenze acquisite

Durante l'esperienza di stage ho appreso diverse tecnologie e strumenti che ritengo mi risulteranno utili anche in futuro.

Payara [15] è sicuramente stato per me lo strumento che ha richiesto più tempo per apprenderlo a pieno, Payara infatti è un server estremamente configurabile sia ad esempio nei log degli applicativi, sia nelle librerie Java [7] che si possono intregare e altrettante funzionalità sul monitoraggio degli applicativi e sulle risorse condivise. Ho migliorato inoltre: la mia conoscenza e padronanza sull'utilizzo dei thread[g], la gestione di connsessioni TCP/IP attraverso l'utilizzo dei socket channels e la capacità di scrivere e classificare i log di sistema.

La necessità di ristrutturare i dati secondo le specifiche del cliente è stata sicuramente istruttiva e ha dimostrato le potenzialità della libreria Gson [6] di Google per la serializzazione di oggetti in JSON.

Ho maturato una discreta esperienza con il protocollo di autorizzazione OAuth 2.0 che è ad oggi il protocollo standard che consente alle applicazioni di accedere a risorse protette di un servizio per conto di un utente.

6.6 Valutazione personale

L'esperienza di stage è stata particolarmente istruttiva e mi ha permesso di avere una prima visione del mondo lavorativo.

Rispetto all'ambiente accademico è risultata un'esperienza più pragmatica e meno legata al rispetto di regole e metodi formali. Ritengo utile sottolineare che le conoscenze da me maturate durante il percorso di laurea si sono rivelate particolarmente utili ad affrontare le sfide che mi sono state proposte durante lo sviluppo e la progettazione richiesta dall'azienda.

Aver gestito per tutta la durata dello stage le comunicazioni tecniche con il cliente finale mi ha permesso di responsabilizzarmi in fretta e di vivere un'esperienza lavorativa reale.

Nel complesso mi ritengo soddisfatto del lavoro svolto e dell'opportunità che mi è stata concessa durante il percorso di Laurea di confrontarmi con il mondo del lavoro.

Acronimi e abbreviazioni

API Application Program Interface. 11, 12, 14, 41, 43

cURL client URL. 2, 29, 32

ITS intelligent transportation system. 1

JSON JavaScript Object Notation. 7, 11, 13, 17–19, 23, 24, 29–32, 40–42

JVM Java Virtual Machine. 7

RDBMS Relational Database Management System. 8

REST Representational state transfer. 2

UML Unified Modeling Language. xi, 18, 22, 25

UUID Universally unique identifier. 18, 30

Glossario

application server in informatica un application server è una tipologia di server che fornisce l'infrastruttura e le funzionalità logiche di supporto, sviluppo ed esecuzione di applicazioni nonché altri componenti server in un contesto distribuito. 6

backend in informatica col termine backend ci si riferisce alla parte non visibile dell'applicazione che gestisce la logica di dominio. 2

curl cURL è un progetto software che predispone sia una libreria che un'interfaccia a riga di comando per trasferire dati utilizzando diversi protocolli di rete. 43

endpoint in informatica col termine endpoint ci si riferisce all'inferfaccia esposta da un'altra macchina a cui si accede tramite un canale di comunicazione. 2, 8, 12–14, 23, 24, 29, 32, 35, 36, 39, 40

firmware Il firmware è un programma, ovvero una sequenza di istruzioni, integrato direttamente in un componente elettronico programmato. 21, 31

framework in informatica e specificamente nello sviluppo software, un framework è un'architettura logica di supporto sulla quale un software può essere progettato e realizzato, spesso facilitandone lo sviluppo da parte del programmatore. 2, 26

frontend in informatica col termine frontend ci si riferisce alla parte visibile dell'applicazione esposta all'utente. 2

intelligent transportation system con ITS o "sistemi di trasporto intelligenti", s'intende l'integrazione delle conoscenze nel campo delle telecomunicazioni, elettronica, informatica - in breve, la "telematica" - con l'ingegneria dei trasporti, per la pianificazione, progettazione, esercizio, manutenzione e gestione dei sistemi di trasporto. 43

Java Virtual Machine la Java virtual machine esegue istruzioni generate da un compilatore Java. Essa consiste in un interpreter bytecode e tempo di esecuzione che consente di eseguire i file di classe Java su qualsiasi piattaforma, indipendentemente dalla piattaforma su cui sono stati sviluppati in origine. 43

JavaScript Object Notation nell'ambito della programmazione web, JSON, acronimo di JavaScript Object Notation, è un formato adatto all'interscambio di dati fra applicazioni client/server. È basato sul linguaggio JavaScript Standard ECMA-262 3ª edizione, ma ne è indipendente. 43

Relational Database Management System In informatica, un sistema di gestione di basi di dati relazionali, noto con l'inglese relational database management system o RDBMS è la tipologia di database management system basata sul modello relazionale, introdotta da Edgar F. Codd negli anni 1970. 43

Representational state transfer Representational state transfer (REST) è uno stile architetturale per sistemi distribuiti. L'espressione "representational state transfer" e il suo acronimo, REST, fu introdotto nel 2000 nella tesi di dottorato di Roy Fielding, uno dei principali autori delle specifiche dell'HyperText Transfer Protocol (HTTP), e vennero rapidamente adottati dalla comunità di sviluppatori Internet. 43

thread Un thread o thread di esecuzione, in informatica, è una suddivisione di un processo in due o più filoni o sottoprocessi che vengono eseguiti concorrentemente da un sistema di elaborazione monoprocessore o multiprocessore o multicore. 28, 29, 42

Universally unique identifier Lo Universally unique identifier è un identificativo usato nelle infrastrutture software, standardizzato dalla Open Software Foundation come parte di un ambiente distribuito di computazione. 43

Bibliografia

Siti web consultati

[1] *Enel X*. URL: https://www.enelx.com/it/it.

[2] *Git*. URL: https://git-scm.com/.

[3] *GitAhead*. URL: https://gitahead.github.io/gitahead.com/.

[4] *Google Style Guides*. URL: https://google.github.io/styleguide/javaguide.html.

[5] *Gson*. URL: https://github.com/google/gson/blob/master/UserGuide.md.

[6] *Gson*. URL: https://github.com/google/gson.

[7] *Java*. URL: https://www.java.com/it/.

[8] *JUnit*. URL: https://junit.org/junit5/.

[9] *La Semaforica S.r.l.* URL: https://www.lasemaforica.com/.

[10] *Mashery*. URL: https://www.tibco.com/it/products/api-management/.

[11] *Mockito*. URL: https://site.mockito.org/.

[12] *MySQL*. URL: https://www.mysql.com/it/.

[13] *Netbeans*. URL: https://netbeans.apache.org/.

[14] *Oauth2 protocol*. URL: https://oauth.net/2/.

[15] *Payara*. URL: https://www.payara.fish/.

[16] *Redmine*. URL: https://www.redmine.org/projects/redmine.

[17] *TEC Systems Engineering S.r.l.* URL: https://www.tecsen.it/.

[18] *Ubuntu*. URL: https://www.ubuntu-it.org/.

[19] *Webmin*. URL: https://www.webmin.com/.

[20] *Windows*. URL: https://www.microsoft.com/it-it/windows.