UNIVERSITÀ DI PISA
Dipartimento Informatica
Corso di laurea triennale in "Informatica"

TIROCINIO: COLLOQUIO TELEMATICO DI AMMINISTRAZIONI PUBBLICHE E TESORIERE CON SIOPE+

Tutore Accademico
Prof. Laura Semini

Tutore Esterno
Rocco De Marco

Il Candidato
Gianmaria di Rito

Anno Accademico 2019/2020

Indice

Capitolo 1

Introduzione

1.1 Ambito del tirocinio

Il tirocinio da me scelto si è svolto presso gli uffici della Golem Software s.r.l.; precisamente in quello situato nella città di Palmi in provincia di Reggio Calabria. Questa società si occupa di fornire soluzioni gestionali per le P.A. e cioè per Enti del comparto pubblico. La mia permanenza nei suddetti uffici è durata all'incirca un mese e 10 giorni; in particolare dal 23 gennaio 2020 fino al 28 febbraio dello stesso anno.

Allo scopo di spiegare l'argomento trattato nel mio praticantato, c'è bisogno di chiarire il contesto in cui si va a porre il modulo software da me realizzato. Innanzitutto, bisogna effettuare un digressione su quella che è la piattaforma SIOPE, le entità che interagiscono con essa e i flussi che gestisce, che, insieme, costituiscono il cuore pulsante del circuito di informazioni di cui mi sono occupato.

1.2 SIOPE+

Il SIOPE+ (Sistema informativo sulle operazioni degli enti pubblici) è un sistema di rilevazione telematica degli incassi e dei pagamenti effettuati dai tesorieri e cassieri di tutte le PA, che nasce dalla collaborazione tra Ragioneria Generale dello Stato, la Banca d'Italia e l'ISTAT, in attuazione dell'articolo 28 della legge n.289/2002, disciplinato dall'articolo 14, commi dal 6 all'11, della legge n. 196 del 2009.

1.3 Ordinativi informatici di Pagamento e di Incasso

Con l'obiettivo di automatizzare il monitoraggio dei pagamenti e degli incassi delle PA, l'art 1, comma 533, della legge 11 dicembre 2016, n. 232, ha reso obbligatorio l'utilizzo degli ordinativi elettronici, «[...]emessi secondo lo standard Ordinativo Informatico emanato dall'Agenzia per l'Italia digitale (AgID)[...]», e trasmessi alle banche tesoriere o cassiere per il tramite della piattaforma SIOPE+, gestita dalla Ragioneria Generale dello Stato e dalla Banca d'Italia. Il flusso dati di SIOPE+ è quindi costituito da ordinativi elettronici che di seguito saranno chiamati: ordinativi informatici di pagamento e di incasso (OPI). Questi OPI prendono il posto di quelli cartacei e sostituiscono l'ordinativo informatico locale (OLI). Essi sono scambiati tra gli Enti del comparto pubblico (di seguito: "Ente" oppure "PA") e le banche tesoriere o cassiere (di seguito: "BT") passando della piattaforma SIOPE+. Il complesso di uno o più ordinativi informatici viene per brevità definito «flusso».

1.4 Flusso di informazioni tra Ente, BT e SIOPE+

Come indicato in precedenza , le P.A. soggette all'art. 1, comma 533, della legge 11 dicembre 2016, n. 232 devono:

1. ordinare incassi e pagamenti alla BT utilizzando esclusivamente ordinativi informatici emessi secondo lo standard definito dall'AgID;

2. trasmettere gli ordinativi informatici alla BT solo ed esclusivamente per il tramite dell'infrastruttura SIOPE+, gestite dalla Banca d'Italia.

SIOPE+ consente di acquisire informazioni dagli Enti in modo automatico, liberando gli stessi dall'obbligo di provvedere alla trasmissione alla Piattaforma elettronica dei Crediti Commerciali (di seguito: sistema PCC) di dati riguardanti il pagamento delle fatture, che costituisce la principale criticità dell'attuale sistema di monitoraggio dei debiti commerciali e dei relativi tempi di pagamento.

Come indicato in Figura 1 : Il processo si attiva dal Sistema di Interscambio (SdI) che invia le fatture elettroniche sia agli Enti pubblici, sia al sistema Piattaforma dei Crediti Commerciali (PCC). L'ente, a sua volta, invia il flusso contenente gli ordinativi, compresi quelli per il pagamento delle

fatture, alla piattaforma SIOPE+, che lo mette a disposizione della BT e aggiorna in automatico il sistema PCC. La BT esegue gli ordinativi contenuti nel flusso e comunica all'Ente l'esito dei singoli ordinativi attraverso la piattaforma SIOPE+.

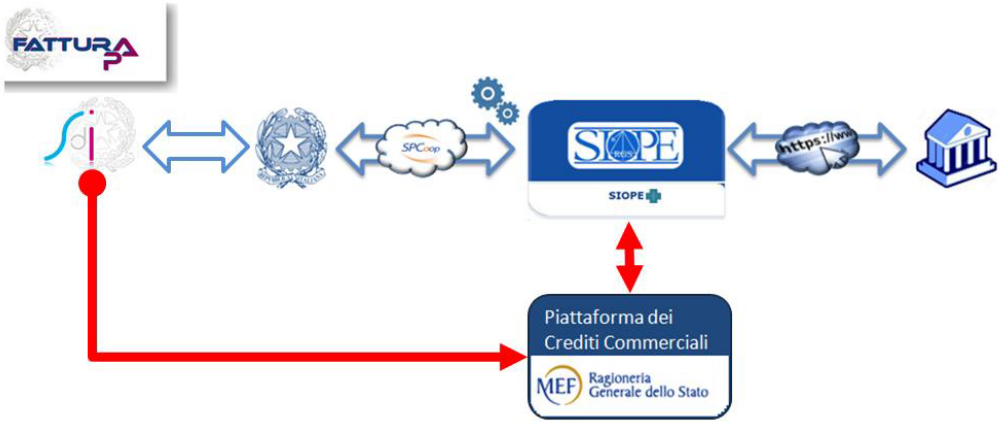

Figura1. Il colloquio tra Ente e BT e il monitoraggio dei crediti commerciali

1.5 Il funzionamento di SIOPE+

Ora che sono state chiarite quali sono le entità in gioco e perché sono collegate tra di loro, possiamo entrare più nello specifico andando a indicare che SIOPE+ supporta esclusivamente una modalità di comunicazione con gli Enti e le BT di tipo Application-to-Application (A2A). Questo, è un modello per l'integrazione diretta tra applicazioni informatiche, ovvero senza la necessaria interazione di un essere umano. Gli Operatori (P.A. e BT) possono demandare a un soggetto terzo (i.e. Tramite PA, Tramite BT) l'implementazione del colloquio tecnico A2A con SIOPE+.

L'interfaccia di SIOPE+ offre agli Operatori servizi utili per l'esecuzione delle seguenti operazioni:

- upload di messaggi;
- download di messaggi;
- inquiry (richiesta lista di link a messaggi che soddisfano determinati criteri di ricerca);

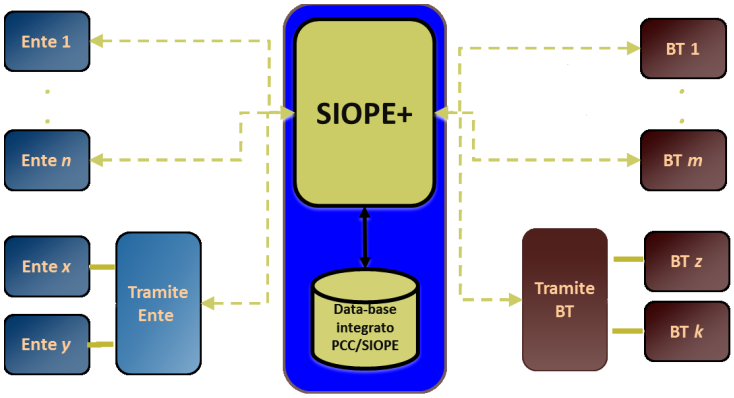

Figura2. Modello concettuale di funzionamento di SIOPE+

1.6 Argomento del Tirocinio

Tornando al punto di partenza, posso ora spiegare che l'oggetto del mio tirocinio è stato appunto, quello di realizzare un modulo che si interponga tra l'Ente e SIOPE+ quindi utile alle P.A. e offerto da quei soggetti che sopra abbiamo definito come Tramite degli Enti. Questa piattaforma ha il compito di fornire un' interfaccia web-based alle PA.

La suddetta offrirà due macro funzionalità:

- La prima permetterà l'interazione delle P.A. con SIOPE+ attraverso vari servizi di tipo REST offerti da SIOPE+ che il nostro sistema avrà il compito di contattare nelle modalità esplicate all'interno del documento [10] in vece dei suddetti Enti, così da scaricare le responsabilità di questi ultimi riguardanti la comunicazione di tipologia A2A.

- La seconda avrà il compito di intercettare, spacchettare e archiviare in un database tutti i flussi che passano presso la nostra piattaforma e quindi che vanno dagli Enti verso SIOPE+ e viceversa, allo scopo di offrire all'utente la possibilità di interrogare il sistema riguardo i flussi inviati e ricevuti.

1.7 Messaggi

Per capire meglio cosa si intende per flussi, di seguito si elencano tutti i messaggi che il protocollo OPI prevede; ricordando però che i suddetti che interessano il nostro progetto sono solo quelli scambiati tra Ente e SIOPE+. Sono stati inseriti tutti per facilitare la creazione di una visione di insieme che altresì sarebbe stata troncata :

- **Flusso Ordinativi**: messaggio generato ed inviato dalla PA a SIOPE+ contenente una lista di ordinativi (mandati/reversali) da far eseguire alla BT.

- **Ricezione Flusso**: messaggio generato ed inviato dalla BT a SIOPE+ per confermare la ricezione di un Flusso Ordinativi. Tale messaggio è in alternativa esclusiva rispetto a quello di Rifiuto Flusso (viene inviato l'uno oppure l'altro).

- **Rifiuto Flusso**: messaggio generato ed inviato dalla BT a SIOPE+ per comunicare la rilevazione di anomalie nel relativo Flusso Ordinativi, che ne determinano il rifiuto. Tale messaggio è in alternativa esclusiva rispetto a quello di Ricezione Flusso (viene inviato l'uno oppure l'altro).

- **Esito Applicativo** (e **Flusso Esiti Applicativi**): messaggio generato ed inviato dalla BT a SIOPE+ per comunicare l'esito dei controlli di merito e l'esito dell'operazione disposta dal singolo ordinativo (e.g. "acquisito", "non acquisito", "pagato", regolarizzato", "non eseguibile"). La BT può aggregare più esiti applicativi in un unico messaggio XML, formando un "Flusso Esiti Applicativi".

- **Flusso Giornale di Cassa**: messaggio generato ed inviato dalla BT a SIOPE+ contenente la lista dei movimenti (e.g. "mandato", "reversale", "giroconto", "anticipazione, "fondo di cassa") operati dalla BT sul conto della PA nel periodo di riferimento. Tipicamente la BT produce ed invia il Giornale di Cassa come rendicontazione di fine giornata.

- **Flusso Disponibilità Liquide**: messaggio generato ed inviato dalla BT a SIOPE+ contenente il saldo del conto corrente di tesoreria e di eventuali altri conti correnti e di deposito della PA nel periodo di riferimento. Tipicamente la BT produce ed invia il Flusso Disponibilità Liquide come rendicontazione mensile.

- **ACK Flusso Ordinativi**: messaggio generato e messo da SIOPE+ a disposizione dell'Ente, contenente il riscontro al messaggio di tipo "Flusso Ordinativi". Può contenere degli elementi di tipo "errore" o "warning" per segnalare anomalie rilevate da SIOPE+

- **ACK Ricezione Flusso**: messaggio generato e messo da SIOPE+ a disposizione della BT, contenente il riscontro al messaggio di tipo "Ricezione Flusso". Può contenere degli elementi di tipo "errore" o "warning" per segnalare anomalie rilevate da SIOPE+. Tale messaggio

è in alternativa esclusiva rispetto a quello di ACK Rifiuto Flusso (viene generato l'uno oppure l'altro).

- **ACK Rifiuto Flusso**: messaggio generato e messo da SIOPE+ a disposizione della BT, contenente il riscontro al messaggio di tipo "Rifiuto Flusso". Può contenere degli elementi di tipo "errore" o "warning" per segnalare anomalie rilevate da SIOPE+. Tale messaggio è in alternativa esclusiva rispetto a quello di ACK Ricezione Flusso (viene generato l'uno oppure l'altro).

- **ACK Esito Applicativo**: messaggio generato e messo da SIOPE+ a disposizione della BT, contenente il riscontro al messaggio di tipo "Esito Applicativo".

- **ACK Giornale di Cassa**: messaggio generato e messo da SIOPE+ a disposizione della BT, contenente il riscontro al messaggio di tipo "Flusso Giornale di Cassa".

- **ACK Disponibilità Liquide**: messaggio generato e messo da SIOPE+ a disposizione della BT, contenente il riscontro al messaggio di tipo "Flusso Disponibilità Liquide".

1.8 Sequenza di colloquio Ente - SIOPE+ - BT

1. L'Ente predispone il messaggio "Flusso Ordinativi" (sottoscritto con firma digitale da soggetto legittimato presso la PA), lo comprime con l'algoritmo ZIP (estensione ".zip") e lo invia a SIOPE+ invocando il relativo servizio di richiesta upload di messaggio su SIOPE+.

2. SIOPE+ può scartare o accettare la richiesta di upload dell'Ente; in caso di accettazione della richiesta SIOPE+ mette a disposizione dell'Ente un messaggio di ACK che conferma la ricezione del "Flusso Ordinativi".

3. L'Ente acquisice il messaggio di ACK invocando il relativo servizio di download di messaggio offerto dall'interfaccia di SIOPE+.

4. SIOPE+ mette a disposizione della BT il messaggio "Flusso Ordinativi" che lo acquisisce invocando il relativo servizio di download offerto dall'interfaccia di SIOPE+.

5. La BT predispone il messaggio "Ricezione Flusso" ovvero "Rifiuto Flusso", lo comprime con algoritmo ZIP e lo invia a SIOPE+ invocando il relativo servizio di richiesta upload di messaggio su SIOPE+.

6. SIOPE+ può scartare o accettare la richiesta di upload della BT; in caso di accettazione della richiesta SIOPE+ mette a disposizione della BT un messaggio di ACK che conferma la ricezione della "Ricezione Flusso"/"Rifiuto Flusso".

7. La BT acquisice il messaggio di ACK invocando il relativo servizio di download1 di messaggio offerto dall'interfaccia di SIOPE+.

8. SIOPE+ mette a disposizione dell'Ente il messaggio "Ricezione Flusso"/"Rifiuto Flusso" che lo acquisisce invocando il relativo servizio di download offerto dall'interfaccia di SIOPE+.

9. La BT predispone il messaggio "Esito Applicativo", lo comprime con algoritmo ZIP e lo invia a SIOPE+ invocando il relativo servizio di richiesta upload di messaggio su SIOPE+.

10. SIOPE+ può scartare o accettare la richiesta di upload della BT; in caso di accettazione della richiesta SIOPE+ mette a disposizione della BT un messaggio di ACK che conferma la ricezione dell'esito applicativo.

11. La BT acquisice il messaggio di ACK invocando il relativo servizio di download1 di messaggio offerto dall'interfaccia di SIOPE+.

12. SIOPE+ mette a disposizione dell'Ente il messaggio "Esito Applicativo" che lo acquisisce invocando il relativo servizio di download offerto dall'interfaccia di SIOPE+.

13. La BT predispone il messaggio "Giornale di Cassa", lo comprime con algoritmo ZIP e lo invia a SIOPE+ invocando il relativo servizio di richiesta upload di messaggio su SIOPE+.

14. SIOPE+ può scartare o accettare la richiesta di upload della BT (cfr. paragrafo 3.6); in caso di accettazione della richiesta SIOPE+ mette a disposizione della BT un messaggio di ACK che conferma la ricezione del "Giornale di Cassa".

15. La BT acquisice il messaggio di ACK invocando il relativo servizio di download1 di messaggio offerto dall'interfaccia di SIOPE+.

16. SIOPE+ mette a disposizione dell'Ente il messaggio "Giornale di Cassa" che lo acquisisce invocando il relativo servizio di download1 offerto dall'interfaccia di SIOPE+.

17. La BT predispone il messaggio “Disponibilità liquide”, lo comprime con algoritmo ZIP e lo invia a SIOPE+ invocando il relativo servizio di richiesta upload di messaggio su SIOPE+.

18. SIOPE+ può scartare o accettare la richiesta di upload della BT; in caso di accettazione della richiesta SIOPE+ mette a disposizione della BT un messaggio di ACK che conferma la ricezione delle “Disponibilità liquide”.

19. La BT acquisice il messaggio di ACK invocando il relativo servizio di download di messaggio offerto dall’interfaccia di SIOPE+.

20. SIOPE+ mette a disposizione dell’Ente il messaggio “Disponibilità liquide” che lo acquisisce invocando il relativo servizio di download offerto dall’interfaccia di SIOPE+.

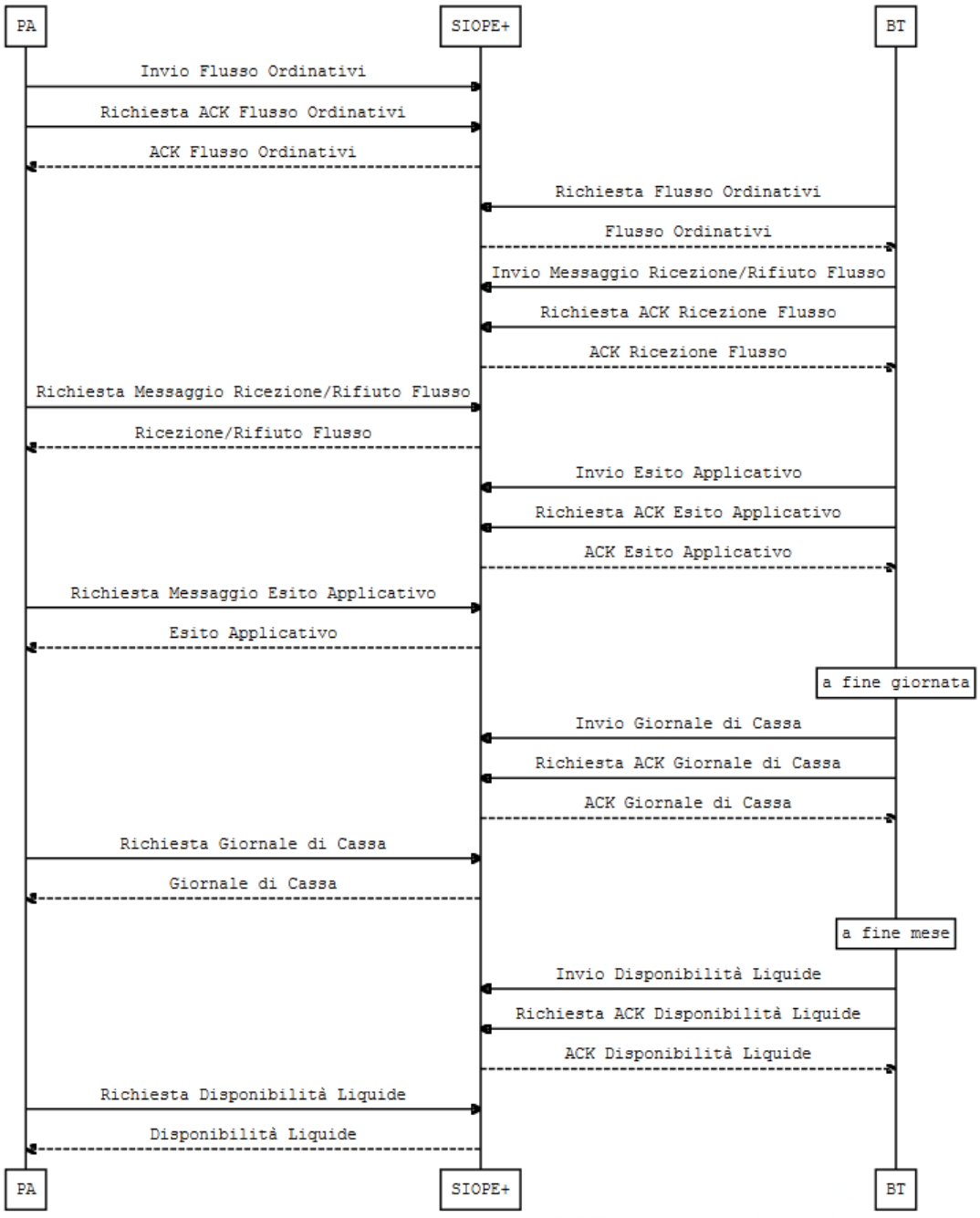

Figura3. Caso esemplificativo di colloquio tra gli Operatori e SIOPE+

Capitolo 2

Tecnologie utilizzate

2.1 Java

Java è un linguaggio di programmazione ad alto livello, orientato agli oggetti e a tipizzazione statica, che si appoggia sull'omonima piattaforma software di esecuzione, specificamente progettato per essere il più possibile indipendente dalla piattaforma hardware di esecuzione. L'utilizzo di questo linguaggio di programmazione è stato imposto dall'azienda proponente del tirocinio attraverso l'utilizzo dell'ambiente di sviluppo Eclipse IDE reperibile nel sito [2]. Di seguito elencherò tutte le librerie native del linguaggio Java che ho utilizzato; spiegando, per quale motivo le ho scelte e per cosa le ho utilizzate.

- **java.io/java.nio** Librerie utili alla manipolazione dei file e alla scrittura/lettura su/da essi. Scelte per la familiarità del candidato nel loro utilizzo. Documentazione reperibile nei siti [3, 4].
- **java.util** Libreria usata per le sue molteplici funzionalità tra cui definizione di varie tipologie di strutture dati. Scelta anch'essa per la familiarità del candidato nell'utilizzarla. Documentazione reperibile al sito [6].
- **java.sql** Libreria utilizzata per implementare la scrittura e la lettura di dati all'interno del database tramite protocollo JDBC. In questo caso la scelta è ricaduta su questa libreria perché per comunicare con il database, utilizza metodi che prendono come argomenti stringhe contenenti query SQL. Data quindi la conoscenza pregressa del candidato nell'utilizzo di query per interagire con i database, si è preferito utilizzare questa soluzione al posto di altre come ad esempio "Hibernate" che propone un approccio diverso per l'accesso ai dati, basato meno sull'invio diretto dei comandi SQL, quanto piuttosto fondato sul mapping

tra sistema informativo relazionale (incluso nel database) e modello a oggetti realizzato nel programma Java. Documentazione reperibile al sito [5].

2.2 Spring Boot

Progetto ideato per fornire funzionalità aggiuntive a Spring che è un framework open source per lo sviluppo di applicazioni su piattaforma Java.

Prima della creazione di Spring Boot, tutte le applicazioni basate su Spring avevano bisogno di un web server, come Tomcat, Jetty o Undertow, per essere eseguite. Con Spring Boot, è possibile creare una applicazione avente un metodo main che lancia l'intera applicazione web, compreso il web server integrato.

Di questo framework ho utilizzato la dipendenza di tipo Web denominata Spring Web utile per implementare servizi REST nelle modalità che vedremo nei capitoli di spiegazione del lavoro svolto. L'utilizzo di questo framework è stato consigliato dal tutore interno all'azienda in correlazione al servizio Sping Inizializr reperibile all'URL [13], che offre la possibilità di creare progetti Spring Boot con le dipendenze scelte già annesse.

Le librerie facenti parte della dipendenza Spring Web che ho utilizzato sono:

- **org.springframework.boot** libreria usata per creare l'ambiente dove sarà eseguito l'applicazione basata su Spring implementata.
- **org.springframework.web** Libreria usata per creare il Client REST realizzato.
- **org.springframework.http** Libreria che offre l'opportunità di creare oggetti come HttpEntity utili anch'essi per l'implementazione del Client REST.

La documentazione relativa a spring boot è reperibile all'URL [12].

Questa è stata la prima occasione per il candidato di usare questa tecnologia e relativamente alle librerie utilizzate ha acquisito una buona dimestichezza nel loro utilizzo.

2.3 Maven

Apache Maven è uno strumento di gestione di progetti software basati su Java e build automation. Per funzionalità è similare ad Apache Ant, ma basato su concetti differenti.Maven usa un costrutto conosciuto come Project

Object Model (POM); un file XML che descrive le dipendenze fra il progetto e le varie versioni di librerie necessarie nonché le dipendenze fra di esse. In questo modo si separano le librerie dalla directory di progetto utilizzando questo file descrittivo per definirne le relazioni. Maven effettua automaticamente il download di librerie Java e plug-in Maven dai vari repository definiti, scaricandoli in locale o in un repository centralizzato lato sviluppo. Questo permette di recuperare in modo uniforme i vari file JAR e di poter spostare il progetto indipendentemente da un ambiente all'altro avendo la sicurezza di utilizzare sempre le stesse versioni delle librerie.

Il suo utilizzo in contemporanea al framework spring sopra definito è stato suggerito dall'azienda. La documentazione relativa a Maven è reperibile al sito [7].

Essendo anche in questo caso il primo utilizzo di questa tecnologia il candidato lo ha utilizzato in larga parte per includere librerie esterne attraverso il file XML POM acquisendo quindi nel suo utilizzo una giusta familiarità.

2.4 SQL Anyware 17 developer

SQL Anywhere 17 è un applicativo che fornisce tecnologie per la gestione e lo scambio di dati, che consente il rapido sviluppo di applicazioni basate su database per ambienti server, desktop e mobili.

Ho utilizzato questo software per realizzare il database relazionale per l'immagazzinamento dei flussi . Il suo utilizzo è stato suggerito dal relatore interno all'azienda.

Altre informazioni riguardanti questo software sono reperibili all'indirizzo [14].

2.5 ODBC

Una fonte dati ODBC (Open DataBase Connectivity) è un file o archivio di database che consente l'accesso ai dati (consultazione e modifica) da parte di un qualsiasi programma Windows. Gli archivi ODBC possono essere interrogati e modificati tramite il linguaggio SQL. Il programma (non database) che ne chiede la consultazione deve essere dotato di un apposito driver ODBC. Questa tecnologia è stata utilizzata per connettere il modulo software realizzato con il database relazionale creato attraverso SQL Anyware.

Come gran parte delle altre tecnologie, il suo utilizzo è stato consigliato dagli sviluppatori della Golem Software. Per approfondire l'argomento, ci si può rivolgere al sito [8]. Nell'utilizzare questo driver ha trovato non poche

difficoltà iniziali a causa della poca familiarità riguardo l'argomento. Col senno di poi però, può affermare di essere riuscito a comprendere il meccanismo dietro il funzionamento dell'ODBC acquisendo quindi le conoscenze necessarie al suo utilizzo in maniera consapevole.

2.6 Postman

Postman è un'applicazione del browser Google Chrome che consente di costruire, testare e documentare API più velocemente. Tramite Postman è possibile effettuare delle chiamate API senza dover mettere mano al codice dell'applicazione, consentendo di effettuare le chiamate tramite questo plugin che fornisce un'utile interfaccia grafica. Le richieste possono essere effettuate sia verso un server locale che verso un server online impostando tutti i dati di una tipica chiamata API, dagli headers al body. Tramite l'interfaccia grafica è possibile selezionare facilmente il tipo di chiamata da effettuate (POST, GET, PUST, DELETE, ecc.), impostare l'url su cui effettuare la chiamata e inviarla.

Questo applicativo è stato utile al candidato per testare i servizi REST offerti da SIOPE+; allo scopo di replicarne il funzionamento all'interno di uno stub che andasse a mimare SIOPE+ stesso, realizzato a scopo di testing. Come in precedenza, il sottoscritto è stato spinto dal proprio tutor interno ad utilizzare questo software. La comprensione del suo funzionamento è stata piuttosto veloce per il sottoscritto, che fin da subito, ha mostrato poca difficoltà nell'utilizzare questa applicazione per adempiere ai suoi scopi. Tutta la documentazione riguardante l'applicativo Postman è consultabile al sito [9].

Capitolo 3

Lavoro svolto

3.1 Studio della documentazione

Dato che il progetto di cui mi sono occupato non era stato trattato in nessun modo, in precedenza, dall'azienda; il tutore aziendale, ha deciso di far cominciare il mio percorso di tirocinio studiando la documentazione. Il primo passo per la realizzazione del modulo software che ho implementato, è stato dunque quello di studiare un gruppo di documenti che costavano delle "Regole tecniche per il colloquio telematico di Amministrazioni pubbliche e Tesorieri con SIOPE+" e delle "Regole tecniche e standard per l'emissione dei documenti informatici relativi alla gestione dei servizi di tesoreria e di cassa degli enti del comparto pubblico attraverso il sistema SIOPE+". I documenti sopra citati, sono reperibili entrambi all'indirizzo [11]. Questa operazione è durata circa due giorni in cui sono stato a stretto contatto con il tutore interno all'azienda allo scopo di comprendere perfettamente, sia il contesto in cui si sarebbe dovuto inserire il software che sarei poi andato a sviluppare, che il funzionamento interno della piattaforma SIOPE+. Quest'ultima è stata la parte più importante da capire, dato che è poi servita sia per costruire un server che offrisse gli stessi servizi di SIOPE+ allo scopo di effettuare testing, che per comprendere al meglio come contattare SIOPE+ nelle giuste modalità. Questa fase è servita inoltre, per ricavare le classi che sarei poi andato a sviluppare e ad identificare le entità che avrebbero in seguito fatto parte del modello E-R alla base del database che ho poi costruito.

3.2 Redazione di un documento di analisi dei requisiti

Finito l'apprendimento riguardante la documentazione, mi è stato fornito un file dal tutore che riguardava il modus operandi per la stesura di un'analisi dei requisiti che avrei dovuto redigere a scopo informativo per coloro che, in seguito, avrebbero ripreso in mano il mio lavoro per proseguirlo ed ampliarlo. Questo file proponeva la compilazione di un'analisi dei requisiti più snella e riassuntiva rispetto allo standard che avevo imparato a conoscere nei diversi corsi che ne parlavano durante il percorso universitario. Esso proponeva di ampliare gli argomenti riguardanti :

- Analisi dello scenario
- Analisi sistemi esistenti
- Diagramma di struttura composita
- Modello E-R Entità Relazione
- Elenco requisiti funzionali
- Elenco requisiti non funzionali e/o di sistema
- Dati gestiti
- Rappresentazione Dati Gestiti

Sono partito dall'analisi dello scenario in cui ho cercato di far comprendere il contesto di sviluppo che avevo carpito dai documenti che nella fase iniziale avevo approfondito. Non essendoci, poi, un sistema esistente ho saltato questa parte passando alla compilazione di quella che trattava il modello a struttura composita e il modello Entità-Relazioni. In questa sezione ho inserito un diagramma di struttura composita molto simile a quello che verrà mostrato in seguito che, come il modello E-R, ho realizzato attraverso un software denominato Visual Paradigm utile per la realizzazione di diagrammi UML. Ho poi elencato tutte le entità e le relazioni del suddetto modello, andando a fornire una breve descrizione per meglio identificarle .

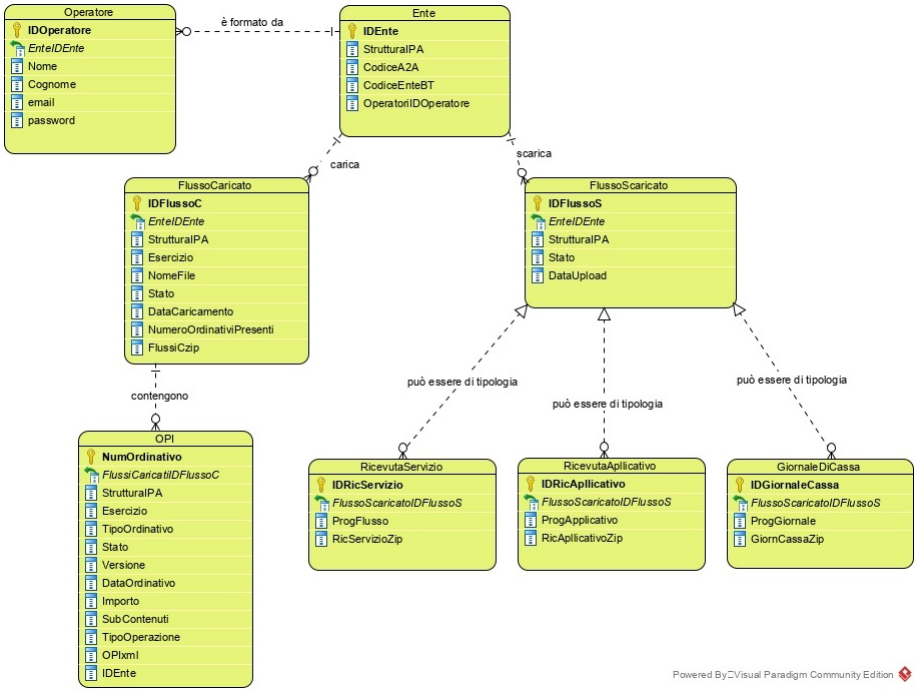

Figura4. Modello concettualizzato

Riguardo i requisiti funzionali, per aiutarmi nel comprendere quali di questi avrei potuto elencare e quindi scegliere di implementare, il tutore mi ha suggerito di visionare altri software web-based che, come quello che sarei andato a realizzare io, svolgono il compito di intermediari per gli Enti che vogliono comunicare con SIOPE+. Uno di quelli che offre più funzionalità è EASYSIOPE che può essere visitato all'indirizzo [1]. Visionando le funzionalità offerte da quest'ultimo e considerando il tempo a mia disposizione per riuscire ad implementare tutti i requisiti funzionali elencati nel documento che stavo redigendo; la lista di requisiti dalla piattaforma verso l'ente è la seguente:

- **Requisito N.1 U1, Registrazione utente;**
  Offre la possibilità ad un utente facente parte di un Ente di iscriversi alla piattaforma

- **Requisito N.2 U2, Login utente;**
  Offre la possibilità ad un utente registrato di accedere alla piattaforma tramite login

- **Requisito N.3 FI1, Importazione Flusso;**
  Offre la possibilità di importate un flusso (insieme di OPI) nella piattaforma

- **Requisito N.4 FI2, Lista Flussi Importati;**
  Mostra la lista dei flussi importati (caricati) in base a vari campi di ricerca

- **Requisito N.5 O1,Lista Ordinativi Ente;**
  Mostra la lista dei singoli OPI lavorati in base a vari campi di ricerca

- **Requisito N.6 FS1,Lista Ricevute di servizio;**
  Mostra la lista delle ricevute di servizio scaricate in base a vari campi di ricerca

- **Requisito N.7 FS2,Lista Ricevute applicativo;**
  Mostra la lista delle ricevute applicativo scaricate in base a vari campi di ricerca

- **Requisito N.8 FS3,Lista Giornale di cassa;**
  Mostra la lista dei giornali di cassa scaricati in base a vari campi di ricerca

Riguardo i requisiti non funzionali l'unica cosa che ho deciso di aggiungere tratta la natura web-based dell'applicativo che stavo andando a sviluppare. Questo implica che il suddetto sarà fruibile solo via web.

Dato che per testare la piattaforma ho realizzato anche uno stub di SIOPE che naturalmente comunica solo con la piattaforma da me sviluppata; ho redatto anche un elenco dei requisiti funzionali che vengono offerti dallo stub di SIOPE alla piattaforma.

- **Requisito N.1 FI1, Importazione Flusso relativo ad uno specifico ente;**
  Offre la possibilità di importate un flusso dalla piattaforma (insieme di OPI) nello stub SIOPE

- **Requisito N.2 L1, Lista Ack Flussi Ordinativi relativa ad uno specifico ente**
  Offre la possibilità di scaricare dallo stub siope la lista dei messaggi di acknolegemet dei flussi ordinativi

- **Requisito N.3 L2,Lista Esiti Flussi Ordinativi relativa ad uno specifico ente;**
  Offre la possibilità di scaricare dallo stub siope la lista dei messaggi di esito flussi ordinativi

- **Requisito N.6 L3,Lista Messaggi Esito Applicativo relativa ad uno specifico ente;**
  Offre la possibilità di scaricare dallo stub siope la lista dei messaggi di esito applicativo

- **Requisito N.7 L4,Lista Giornale di Cassa relativa ad uno specifico ente;**
  Offre la possibilità di scaricare dallo stub siope la lista dei giornali di cassa

- **Requisito N.8 L5,Lista Disponibilità Liquide relativa ad uno specifico ente;**
  Offre la possibilità di scaricare dallo stub siope la lista dei flussi di disponibilità liquide

- **Requisito N.9 D1, Download Ack Flussi Ordinativi relativa ad uno specifico ente**
  Offre la possibilità di scaricare dallo stub siope dei messaggi di acknolegemet dei flussi ordinativi

- **Requisito N.10 D2,Download Esiti Flussi Ordinativi relativa ad uno specifico ente;**
  Offre la possibilità di scaricare dallo stub siope dei messaggi di esito flussi ordinativi

- **Requisito N.11 D3,Download Messaggi Esito Applicativo relativa ad uno specifico ente;**
  Offre la possibilità di scaricare dallo stub siope dei messaggi di esito applicativo

- **Requisito N.12 D4,Download Giornale di Cassa relativa ad uno specifico ente;**
  Offre la possibilità di scaricare dallo stub siope dei giornali di cassa

- **Requisito N.13 D5,Download Disponibilità Liquide relativa ad uno specifico ente;**
  Offre la possibilità di scaricare dallo stub siope dei flussi di disponibilità liquide

Infine ho stilato un elenco dei dati gestiti nelle entità del modello E-R elencandone tutti gli attributi con relativi constraint (Primary Key, Foreign Key) e tipologie di dominio (INT, CHAR, DATE, ecc...).

La stesura di questo documento è durata all'incirca 3 giorni in cui ho chiesto indicazioni sul da farsi al tutore che mi ha aiutato a proseguire ogniqualvolta mi bloccavo su qualche punto del documento. Questa fase è terminata con la lettura e l'approvazione di quanto da me stilato da parte del tutore stesso.

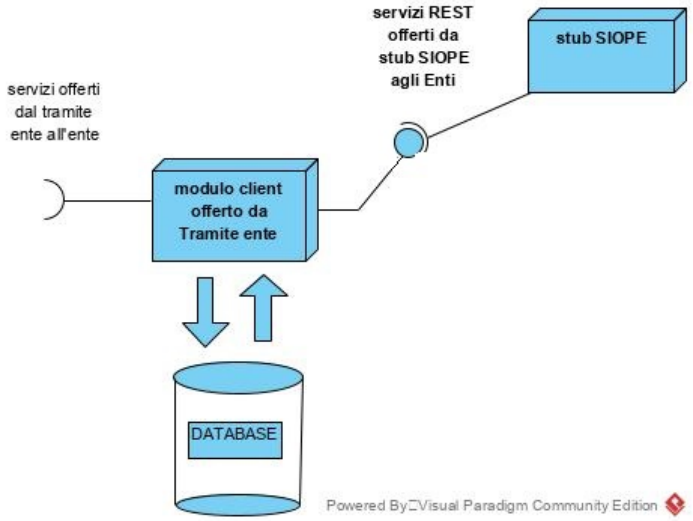

Figura5. Mostra la struttura dei componenti che identifica il sistema sviluppato

3.3 Implementazione Client e Server REST

Terminata la fase descritta sopra, sono passato all'implementazione vera e propria del codice.

La prima cosa che ho fatto in questa parte del tirocinio è stato creare due progetti Maven denominati "Client" e "SIOPE" rispettivamente client e server attraverso il servizio Spring Boot initializr reperibile al sito [13].

In essi, al momento della creazione, ho incluso solo la dipendenza al pacchetto spring web utile per implementare servizi RESTful.

Ho poi importato i progetti nella piattaforma Eclipse IDE attraverso il comando File ->Import -> Maven Project.

Il framework Spring Boot, come spiegato nella parte relativa alle tecnologie utilizzate, offre, un metodo main che lancia l'intera applicazione web, compreso il web server integrato.

Ecco uno spezzone di codice che rappresenta un metodo main creato attraverso Spring Boot.

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class SiopeApplication {
 public static void main(String[] args) {
  SpringApplication.run(SiopeApplication.class, args);
 }
}
```

3.4 Progetto SIOPE

Dopo aver creato i due progetti, mi sono concentrato sull'implementazione di quello che avrebbe avuto il compito di mimare il comportamento di SIOPE+ allo scopo di testare il client che identificava il vero modulo software che mi era stato chiesto di realizzare per il mio progetto di tirocinio.

Come già detto questo modulo server è stato nominato SIOPE; esso ha il compito di fornire tutti i servizi REST che la vera piattaforma SIOPE+ avrebbe fornito ad un Ente nel caso in cui quest'ultimo avesse provato ad interagire con essa.

Innanzitutto, per spiegare l'implementazione di questo componente software c'è bisogno di fare una piccola digressione sulle modalità di utilizzo delle librerie di Spring Boot usate. Esse sono elencate nella parte relativa alle tecnologie utilizzate e hanno un interessate metodo di utilizzo. Infatti, oltre a comprendere metodi da richiamare, esse riconoscono delle annotazioni della forma "@ParolaChiave" per l'identificazione di alcuni tipi come determinati componenti. Queste annotazioni devono essere poste direttamente sopra i tipi che si vogliono identificare. Un esempio di utilizzo è portato di seguito.

```
@RestController
public class SiopeController{....
```

In questo caso si voleva identificare la Classe SiopeController come un componente Controller e cioè, come lo strato MVC della nostra web application. Un Controller Spring è un' entità legata ad un server REST in cui si definiscono tutti i servizi che il server stesso offre sotto forma di metodi.

Possiamo quindi proseguire andando a specificare le varie classi che compongono il progetto SIOPE saltando la classe **SIOPEApplication.java** dato che contiene unicamente il metodo main del progetto creato attraverso Spring Boot Initializr.

3.4.1 SIOPEController

Questa classe identifica il Controller del nostro server REST. In essa sono quindi implementati i metodi che andranno a fornire i servizi che il server offrirà. Essi sono in tutto 11 ma possiamo riassumerli in 3 tipologie principali. Queste sono:

1. servizi di tipo POST;
2. servizi di tipo GET che restituiscono una lista di oggetti;
3. servizi di tipo GET che restituiscono un file .zip.

Della prima tipologia fa parte un unico servizio di cui di seguito riporto il codice del metodo

```
@RequestMapping(value = "v1/{idA2A}/PA/{codEnte}/flusso/",
                method = RequestMethod.POST,
                produces = "application/json;charset=UTF-8",
                consumes= "application/zip" )
public UploadFlussoOrdinativi  UploadFlusso(
                               @PathVariable("idA2A") String idA2A,
```

```
                                    @PathVariable("codEnte") Integer codEnte,
                                    @RequestBody byte[] request
                                    HttpServletResponse response) throws Exception {


HttpEntity<byte[]> r = new HttpEntity<byte[]>(request);
createZipFile(r.getBody());
String fileDownloadUri =ServletUriComponentsBuilder.
                        fromCurrentContextPath().
                        path("v1/{idA2A}/PA/{codEnte}/flusso/{progFlusso}/").
                        path("flussoUpload").toUriString();

response.addDateHeader("Date", Timestamp.valueOf(getDate()).getTime());
response.setContentType("application/json;charset=UTF-8");
response.setHeader("location",fileDownloadUri);

return new UploadFlussoOrdinativi(newRandom().nextInt(1000000000),
                                  getDate(),false,fileDownloadUri) ;
}
```

Inizio il commento del codice dicendo che il metodo è denominato "Upload-Flusso" e offre la possibilità di caricare file di tipo zip (contenenti file XML) sul server. Posto subito sopra di esso c'è l'annotazione "@RequestMapping" che permette l'identificazione di varie informazioni come l'URL a cui è reperibile il metodo, la tipologia del metodo (GET, POST, PUT, ecc...), il tipo di dato che restituisce e il tipo di dato che prende come parametro.

Inoltre, possiamo vedere che, nell'URL a cui è reperibile il servizio, ci sono due campi tra parentesi graffe cioè "idA2A" e "codEnte". Scrivendoli in questa forma, infatti, è possibile, attraverso il comando "@PathVariable", averli come parametri iniziali del metodo; ad esempio, per effettuare su di essi dei controlli.

Il metodo, oltre a questi ultimi, prende come parametro una variabile denominata "request" che è preceduta dall'annotazione "@RequestBody" il quale indica che request conterrà il contenuto della richiesta http del client che ha contattato il nostro servizio.

In fine, nei parametri è presente una variabile "response" che verrà utilizzata per settare l'header della risposta http che il nostro server manderà ad un eventuale client che lo contatta.

Passiamo ora al contenuto del metodo. La prima cosa che viene fatta è l'estrazione del body dalla variabile "request" attraverso la creazione di una variabile di tipo HttpEntity che permette appunto di ricavare sia l'header che

il body di una richiesta. Questa operazione ci è utile per passare al metodo ausiliario "createZipFile" proprio il body della richiesta che conterrà un file zip in formato byte[]. Questo metodo ausiliario andrà a creare concretamente il file zip caricato dal client che ha contattato il servizio, all'interno del server.

Le successive tre righe di codice si occupano di andare a configurare l'header della risposta http che il server farà reperire al client attraverso il settaggio della variabile "response".

Prima di commentare l'ultima riga di codice bisogna specificare che il protocollo REST (Representational State Transfer) realizza un'architettura client-server di tipo STATELESS per scambiare dati utilizzando il protocollo HTTP. Il protocollo HTTP REST, in pratica, non è altro che l'estensione di una SOA (Service-Oriented Architecture) basata su messaggi SOAP, però al contrario di SOAP, i messaggi scambiabili prevedono quattro metodi di trasmissione a differenza delle sole GET e POST delle comuni SOA. Infatti i metodi di trasmissione in un REST Controller Spring sono GET, POST, PUT e DELETE. Si tratta quindi di una SOA RESTFull e dato che esse accettano come messaggio inviato sia XML sia JSON; tutti i messaggi di risposta del nostro server saranno mandati nel formato JSON. Utilizzando messaggi JSON, ci sarà bisogno di serializzarli e deserializzarli per il loro invio e ricezione. Bisognerà dunque creare per ogni tipologia di risposta che il nostro server può mandare, una classe ad hoc che la istanzierà per rendere possibile la successiva serializzazione. Queste classi saranno utilizzate :

- dai metodi della classe "SIOPEController" per creare delle istanze dei messaggi che il protocollo REST serializerà sotto forma di JSON e che verranno spediti al client che ha contattato il servizio.
- dal client per deserializzare le risposte del server arrivate sotto forma di JSON.

Nell'ultima riga di codice quindi, il nostro metodo "UploadFlusso", restituisce un'istanza della classe "UploadFlussoOrdinaivo" la quale istanzia la risposta che questo servizio dovrebbe restituire al client che lo ha contattato.

Della seconda categoria di metodi; che ricordiamo essere i servizi di tipo GET che restituiscono una lista di oggetti; fanno invece parte 5 metodi :

1. **ListaAckFlussoOrdinativi;**
2. **ListaEsitiFlussoOrdinativi;**
3. **ListaMessaggiEsitoApplicativo;**
4. **ListaGiornaleDiCassa;**

5. **ListaDisponibilitàLiquide.**

Essi differiscono oltre che per l'URL a cui reperirli, solo dalla classe che viene utilizzata per istanziare il messaggio di risposta da spedire al client che ha contattato il servizio. Si mostra quindi solo uno dei 5 metodi per evitare ripetizioni inutili.

```
@RequestMapping( value = {
                  "v1/{idA2A}/PA/{codEnte}/flusso/ack/?",
                  "v1/{idA2A}/PA/{codEnte}/flusso/ack/?pagina={pagina}",
                  "v1/{idA2A}/PA/{codEnte}/flusso/ack/?download={download}",...
                  },
                  method = RequestMethod.GET,
                  produces = "application/json;charset=UTF-8")
public ListaAckFlussoOrdinativi ListaAckFlussoOrdinativi(
                        @PathVariable("idA2A") String idA2A,
                        @PathVariable("codEnte") Integer codEnte,
                        @PathVariable(required = false) String dataProduzioneDa,
                        @PathVariable(required = false) String dataProduzioneA,
                        @PathVariable(required = false) String download,
                        @PathVariable(required = false) Integer pagina,
                        HttpServletResponse response
                        ) {

    response.setContentType("application/json;charset=UTF-8");
    response.addDateHeader("Date", Timestamp.valueOf(getDate()).getTime());
    response.setStatus(HttpServletResponse.SC_OK);

    List<infoAck> l = new  ArrayList<infoAck>();
    l.add(new infoAck(124567892,"2016-12-09T11:22:34.000",false,"rhwrretjhrg"));
    l.add(new infoAck(1234567893,"2016-12-09T22:33:44.000",false,"herjrty"));
    return new ListaAckFlussoOrdinativi (12345,124,100, 1,
                                         "2016-12-09T11:22:33.444",
                                         "2016-12-09T22:33:44.555",
                                         l);
     }
```

Iniziamo il commento del codice indicando che questa famiglia di metodi offre come risposta una lista di informazioni creata in base a dei parametri di ricerca che il client può inviare all'interno dell'URL a cui è reperibile il servizio REST. Questi parametri sono opzionali e vengono inseriti nella

porzione di URL che segue il "?". Se tale porzione non viene specificata, si intende richiedere una lista di risultati completa; in caso contrario, i vari parametri possono essere combinati a piacimento per filtrare di conseguenza la lista di risultati.

I parametri di ricerca sono :

- dataProduzioneDa;
- dataProduzioneA;
- download;
- pagina.

A titolo di esempio:

- download=false: tutti i risultati disponibili ma non ancora scaricati;
- dataProduzioneDa=2016-12-09T11:22:33.444 and download=true:
  i soli risultati disponibili a partire dal 2016-12-09T11:22:33.444 e già scaricati;
- dataProduzioneDa=2016-12-09T11:22:33.444 and
  dataProduzioneA=2016-12-09T22:33:44.555:
  i soli risultati disponibili a partire dal 2016-12-09T11:22:33.444 e fino al 2016-12-09T22:33:44.555, già scaricati o non ancora scaricati.

Dato che questa tipologia di metodi ha la necessità di essere reperibile a diversi URL, poiché, come detto sopra, i parametri possono essere cambiati a piacimento per filtrare la lista dei risultati; al campo "value" dell'annotazione RequestMapping, viene assegnata una lista di stringhe. Questa lista conterà tutte le possibili URL creabili attraverso le combinazioni di parametri sopra definiti. Esse non sono tutte visibili nel codice sovrastante dato che avrebbero appesantito troppo la lettura.

Sempre tra i campi dell'annotazione RequestMapping troviamo "method" a cui è assegnato "RequestMethod.GET" utile ad indicare che il metodo è di tipologia GET e "produces" alla quale è assegnato "application/json;charset=UTF-8" utile per indicare che il servizio produce un messaggio di tipo JSON. Passiamo ora ai parametri del metodo che come nel primo servizio sono ricavati attraverso l'annotazione "PathVariable" che in questo caso, per i paramentri ricavati dai campi opzionali dell'URL, è seguita dal settaggio "required = false" il quale farà si che il parametro in questione potrà anche non essere inizializzato nel caso in cui il campo relativo non compare all'interno dell'URL contattata. Tra i paramentri del metodo è poi presente come sempre una variabile "response" per settare l'header della risposta.

Il codice segue andando ad impostare i vari settaggi dell'header della risposta che si sta creando previsti dalle specifiche. Dato che il messaggio di risposta mandato da questa famiglia di metodi contiene una lista, la classe utile ad istanziarlo contiene a sua volta un'ArrayList il cui tipo è un ulteriore classe creata ad hoc; in questo caso denominata "infoAck". Questa ulteriore classe conterrà le informazioni relative al singolo elemento della lista contenuta nel messaggio da recapitare al client che ne ha fatto richiesta.

La Struttura di tutte queste classi create ad hoc sarà discussa ulteriormente in seguito.

Come si può vedere anche nel metodo precedente i valori per l'inizializzazione di queste classi sono casuali questo perché, come già detto, il progetto denominato SIOPE è utile solo allo scopo di testare l'altro progetto denominato Client.

Come per l'altro metodo anche questo restituisce al client che lo ha contattato un'istanza di una classe che in questo caso è "ListaAckFlussoOrdinativi" che verra convertita in JSON dal protocollo REST e poi spacchettata dal client.

Terminiamo il commento delle tre tipologie di metodi implementati all'interno della classe "SIOPEControlle" con i servizi di tipo GET che restituiscono un file .zip.

Anche in questo caso i metodi facenti parte di questa famiglia sono 5:

1. **DownloadAckFlussoOrdinativi;**
2. **DownloadEsitiFlussoOrdinativi;**
3. **DownloadMessaggiEsitoApplicativo;**
4. **DownloadGiornaleDiCassa;**
5. **DownloadDisponibilitàLiquide.**

Come prima riportiamo solo uno dei 5 metodi che anche in questo caso differiscono solo per il contenuto del file zip scaricato e ovviamente per l'URL a cui sono reperibili.

```
@RequestMapping(value = "v1/{idA2A}/PA/{codEnte}/flusso/{progFlusso}/ack",
                method = RequestMethod.GET,
                produces="application/zip")
public byte[] DownloadAckFlusso(
                        @PathVariable("idA2A") String idA2A,
                        @PathVariable("codEnte") Integer codEnte,
```

```
                        @PathVariable("progFlusso") Integer progFlusso,
                        HttpServletResponse response
                        ) throws IOException{
     new Timestamp(10000000);

    response.addDateHeader("Date", Timestamp.valueOf(getDate()).getTime());
    response.setStatus(HttpServletResponse.SC_OK);
    response.addHeader("Content-Disposition", "form-data; name=\"attachment\";
    response.setContentType("application/zip");
    response.setContentLength(10000000);

    ByteArrayOutputStream byteArrayOutputStream = new ByteArrayOutputStream();
    BufferedOutputStream bufferedOutputStream;
    bufferedOutputStream = new BufferedOutputStream(byteArrayOutputStream);
    ZipOutputStream zipOutputStream;
    zipOutputStream = new ZipOutputStream(bufferedOutputStream);

    File file = new File("flusso_opi_importato_18517502.xml");

    zipOutputStream.putNextEntry(new ZipEntry(file.getName()));
    FileInputStream fileInputStream = new FileInputStream(file);

    IOUtils.copy(fileInputStream, zipOutputStream);

    fileInputStream.close();
    zipOutputStream.closeEntry();

     if (zipOutputStream != null) {

        zipOutputStream.finish();
        zipOutputStream.flush();
        IOUtils.closeQuietly(zipOutputStream);

    }

    IOUtils.closeQuietly(bufferedOutputStream);
    IOUtils.closeQuietly(byteArrayOutputStream);
    return byteArrayOutputStream.toByteArray();


}
```

Questo metodo ha il compito di mandare al client che lo ha contattato un file zip sotto forma di byte[ ]. Il file in questione, sarà identificato dal parametro "progFlusso" previsto all'interno dell'URL utile a contattare il servizio. Come per gli altri metodi, anche in questo caso nell'annotazione "RequestMapping" sono inclusi "value" a cui è assegnato l'URL per reperire il servizio, "method" il cui valore setterà questo servizio come uno di tipologia GET e "produces" il cui valore indicherà che il body del messaggio di ritorno conterrà un file nel formato zip. Come si può vedere, per questa famiglia di servizi non è stato necessario costruire delle classi ad hoc per il trasferimento tramite JSON infatti il tipo di dato che restituiscono è un semplice byte[ ]. Questo perché non hanno la necessità di trasferire un dato composto. Come sempre tra i parametri presi in input dal metodo ci sono i vari campi dell'URL reperibili attraverso "PathVariable" e la variabile di tipo "HttpServletResponse" per il settaggio dell'header della risposta. Il contenuto del metodo inizia anche in questo caso con l'aggiunta di vari parametri previsti nelle specifiche all'header del messaggio che si sta costruendo. Si passa poi alla parte del codice che si occupa di costruire il file di tipo zip a partire da un file presente nel server. In realtà il funzionamento della piattaforma SIOPE avrebbe sicuramente previsto una query che prendeva come parametro di ricerca il "progFlusso" prima discusso, per reperire il flusso richiesto all'interno di un database. Ritornando al codice possiamo dire che, prima di tutto, viene creato un "ByteArrayOutputStream" allo scopo di scrivere su di un file. In seguito quest'ultimo viene incapsulato in un "BufferedOutputStream" per permettere la bufferizzazione dei byte che passano nel canale. A sua volta il "BufferedOutputStream" viene poi incapsulato in uno "ZipOutputStream" allo scopo di scrivere su di un file di tipo zip. Viene poi creata una variabile di tipo "File" che identifica il file da mandare ed un file di tipo zip che avrà lo stesso nome del file da mandare e che sarà collegato allo "ZipOutputStream". Segue la creazione di un "FileInputStream" che leggerà sul file da mandare. Il processo si conclude con la copia del contenuto del file da mandare sul nuovo file di tipo zip creato. Questo avviene tramite l'ausilio del metodo statico "copy" previsto dalla classe "IOUtils" contenuta nella librerie di Maven. Il metodo si conclude con una serie di comandi che assicurano la corretta chiusura dei vari canali e con la successiva restituzione del contenuto del "byteArrayOutputStrem" convertito in Byte[ ] che conterà il file richiesto in formato zip.

3.4.2 Classi per il trasferimento tramite JSON

Il progetto denominato SIOPE oltre a contenere la classe "SIOPEApplication" e la classe "SIOPEController", come anticipato nei commenti al codice discusso sopra, contiene anche delle classi create ad hoc per permettere l'invio dei messaggi di risposta sotto forma di JSON. Queste classi possono essere divise in due categorie:

- classi il cui nome è così composto
  "Lista + <nome del metodo che la utilizza>" che istanziano la risposta che viene restituita dal metodo omonimo, contenuto in "SIOPEController", al client che lo ha contattato;

- classi il cui nome è così composto
  "info + <nome del metodo che la utilizza>" che istanziano il singolo nodo della lista presente all'interno della classe omonima dell'altra tipologia.

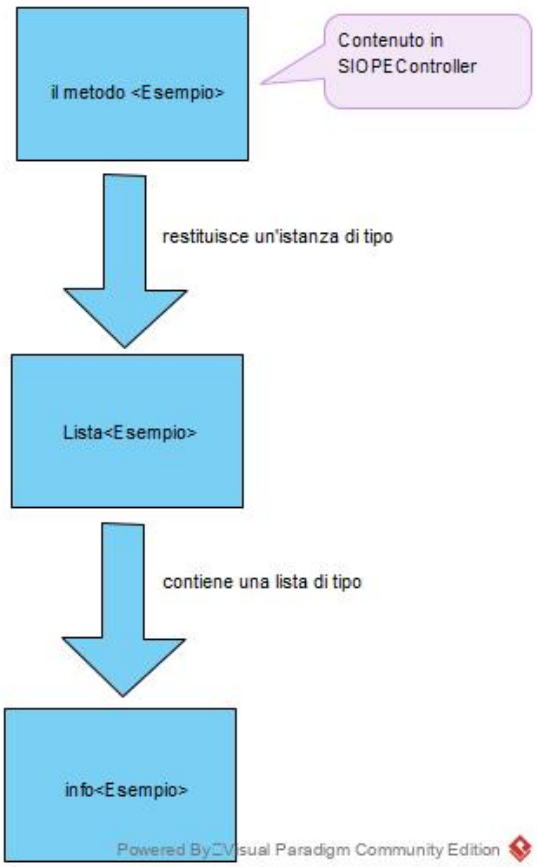

Figura6. Schema di utilizzo delle classi ad hoc

Della prima categoria di classi fanno parte:

1. ListaAckFlussoOrdinativi;
2. ListaDisponibilitaLiquide;
3. ListaEsitiFlussoOrdinativi;
4. ListaGiornaliDiCassa;
5. ListaMessaggiEsitoApplicativo;

Della seconda fanno invece parte:

1. infoAckFlussoOrdinativi;
2. infoDisponibilitaLiquide;
3. infoEsitiFlussoOrdinativi;

4. infoGiornaliDiCassa;

5. infoMessaggiEsitoApplicativo;

Dato che la struttura di queste due categorie di classi è il medesimo a parità di tipologia, riportiamo ora solo la coppia di classi "ListaAckFlussoOrdinativi" e "infoAckFlussoOrdinativi".

```
public class ListaAckFlussoOrdinativi {

  private int numRisultati ;
  private int numPagine ;
  private int risultatiPerPagina;
  private int pagina;
  private String dataProduzioneDa;
  private String dataProduzioneA;
  @JsonProperty("infoAckFlussoOrdinativi")
  private List<infoAckFlussoOrdinativi> l= new ArrayList<>();

  public ListaAckFlussoOrdinativi(int numRisultati, int numPagine,
                                  int risultatiPerPagina, int pagina,
                                  String dataProduzioneDa,
                                  String dataProduzioneA,
                                  List<infoAck> risultati){

        this.numRisultati = numRisultati;
        this.numPagine = numPagine;
        this.risultatiPerPagina = risultatiPerPagina;
        this.pagina = pagina;
        this.dataProduzioneDa = dataProduzioneDa;
        this.dataProduzioneA = dataProduzioneA;
        this.l = risultati;
  }

  public ListaAckFlussoOrdinativi(){}

  public int getNumRisultati() {return numRisultati;}
  public int getNumPagine() {return numPagine;}
  public int getRisultatiPerPagina() {return risultatiPerPagina;}
  public int getPagina() {return pagina;}
  public String getDataProduzioneDa() {return dataProduzioneDa;}
  public String getDataProduzioneA() {return dataProduzioneA;}
```

```
  @Override
  public String toString() {
  String s ="{\n"
  + "numRisultati:" + numRisultati + ",\n"
  +"numPagine:" + numPagine + ",\n"
  + "risultatiPerPagina:" + risultatiPerPagina + ",\n"
  + "pagina:" + pagina + ",\n"
  + "dataProduzioneDa:" + dataProduzioneDa + ",\n"
  + "dataProduzioneA:" + dataProduzioneA + ",\n"
  + "rtisultati: [\n";
  for(int i =0; i < l.size();i++){
      s = s + l.get(i).toString();
      if(l.size() != i) s = s +",";
  }
  s = s + "]";
  return s;
}
```

Questa è una normale classe per il contenimento di dati che prevede due costruttori, uno con parametri e uno di default. Sono presenti poi metodi "getter" per ogni dato contenuto ed un metodo "toString" per la stampa dell'intera classe sotto forma di stringa.

```
public class infoAckFlussoOrdinativi {

   private int progFlusso;
   private String dataProduzione;
   private boolean download ;
   private String location;

   public infoAckFlussoOrdinativi() {}

   public infoAckFlussoOrdinativi( int progFlusso , String dataProduzione ,
                                   boolean download, String location) {
      this.progFlusso = progFlusso ;
      this.dataProduzione = dataProduzione;
      this.download = download;
      this.location = location ;
   }
```

```
    public int getProgFlusso() { return progFlusso;}
    public String getdataProduzione() {return dataProduzione;}
    public boolean getDownload() {return download;}
    public String getLocation() {return location;}

    @Override
    public String toString() {
    return "{" + "progFlusso:'" + progFlusso + '\'' + ",\n"
               + "dataProduzione:'" + dataProduzione + '\''+ ",\n"
               + "download:'" + download + '\''+ ",\n"
               + "location:'" + location + '\''+ ",\n"
               + '}';
    }
}
```

Anche in questo caso siamo dinanzi ad una semplice classe contenitore che come prima ha al suo interno due costruttori, dei metodi "getters" ed un metodo "toString" per la stampa della classe.

Concludo questa sezione sulle classi ad hoc dicendo che esse, fanno parte anche del progetto Client per permettere la deserializzazione dei messaggi JSON che arriveranno come risposta fornita dai vari servizi del progetto SIOPE contattati.

3.5 Progetto Client

Terminata la spiegazione di tutto quello che compone il progetto SIOPE, passiamo ora alla descrizione del vero fulcro del tirocinio e cioè il progetto Maven denominato "Client". Cominciamo elencando tutte le classi che lo compongono per passare poi ad approfondirne il contenuto e le relazioni tra esse.

- **ClientApplication;**
- **IspectorRicevuteApplicatio;**
- **IspectorRicevuteServizio;**
- **IspectorGiornaleCassa;**

- **OpWithDB;**
- **ReaderXML;**
- **Classi per il trasferimento tramite JSON.**

Come "Classi per il trasferimento tramite JSON" si intendono tutte le classi esaminate nell'omonima sottosezione trattata in precedenza, che ovviamente non saranno riesaminate per ovvi motivi. Allo scopo di dare una prima idea su tutte queste classi, diciamo che la classe principale è "ClientApplication" mentre le altre sono classi sue ausiliarie che svolgono vari compiti che esamineremo nello specifico nelle sottosezioni che le riguarderanno. ClientApplication si servirà quindi delle altre classi per operare. Al fine di fornire una spiegazione ben strutturata, partiremo con l'approfondire le classi ausiliarie così da fornire una comprensione chiara della classe principale nelle parti in cui richiamerà le suddette classi.

3.5.1 OpWithDB

Inizieremo con la classe preposta a fare da tramite tra il modulo "Client" e il database relazionale da me realizzato per salvare tutti i dati caricati e scaricati dai vari enti da SIOPE tramite la mia piattaforma. Da quì il nome "OpWithDB". Questa classe funge da contenitore per tutti i metodi creati ad hoc per eseguire delle query specifiche sul database.

```
import java.sql.*;
import java.util.HashMap;

public class OpWithDB {
static  String JDBC_DRIVER;
static  String DB_URL;
static  String USER;
static  String PASS;

public OpWithDB() {
// JDBC driver name and database URL
JDBC_DRIVER = "sap.jdbc4.sqlanywhere.IDriver";
DB_URL = "jdbc:sqlanywhere:DSN=service1";
```

```
//  Database credentials
USER = "gianmaria";
PASS = "golem.123";
}

public String[] CercaOperatore(String IDOperatore,
                               String IDEnte,
                               String password) {
Connection conn = null;
Statement stmt = null;
String Request = "";
String[] s = new String[5];
try{

//STEP 1: Register JDBC driver
Class.forName(JDBC_DRIVER);

//STEP 2: Open a connection
conn = DriverManager.getConnection(DB_URL,USER,PASS);

//STEP 3: Execute a query
stmt = conn.createStatement();
Request = "SELECT o.Nome, o.Cognome, e.IDEnte, e.StrutturaIPA, e.CodiceA2A  "
+ "FROM Operatori AS o INNER JOIN Enti AS e ON  e.IDEnte = o.IDEnte "
+ "WHERE o.IDOperatore="+IDOperatore+" AND o.IDEnte="+IDEnte+" AND
o.password="+password;

ResultSet rs = stmt.executeQuery(Request);
while(rs.next()) {
    s[0] =rs.getString("Nome");
    s[1] = rs.getString("Cognome");
    s[2] = String.valueOf(rs.getInt("IDEnte")) ;
    s[3] = rs.getString("StrutturaIPA");
    s[4] = rs.getString("CodiceA2A");
}

//STEP 4: Clean-up environment
rs.close();
stmt.close();
conn.close();
```

```
}catch(SQLException se){se.printStackTrace(); }
catch(Exception e){e.printStackTrace(); }

finally{

     try{
         if(stmt!=null) stmt.close();
    }catch(SQLException se2){}

     try{
         if(conn!=null) conn.close();
        }catch(SQLException se){ se.printStackTrace(); }

 }
return s;
}

public int GET_IDENTITY(String Tab) {...}

public String INSERT(String Request) {...}

public HashMap<String,byte[]> SELECTFS(String Tab,
                                       String Prog,
                                       String DataUploadDa,
                                       String DataUploadA) {...}


public HashMap<String,byte[]> SELECTFC(String Tab,
                                       String nomefile,
                                       String DataCaricamentoDa,
                                       String DataCaricamentoA,
                                       String esercizio) {...}

public  HashMap<String, byte[]> SELECTOPI(String Tab,
                                          String DataOrdinativoDa,
                                          String DataOrdinativoA,
                                          String esercizio,
                                          String tipoOrdinativo,
                                          String tipoOperazione,
                                          String numOrdinativoDa,
                                          String numOrdinativoA) {...}
```

```
private HashMap<String,byte[]> formaRisultSELECT(String Tab, ResultSet rs)
                               throws SQLException {
                               String s = "";
    HashMap<String,byte[]> ris = new HashMap<String, byte[]>();
    switch(Tab) {

case("FlussiCaricati"):
   while(rs.next()){

    int idFlussoC  = rs.getInt(1);
    int idEnte = rs.getInt(2);
    String StrutturaIPA = rs.getString(3);
    int Esercizio = rs.getInt(4);
    String NomeFile = rs.getString(5);
    String Stato = rs.getString(6);
    String DataCaricamento = rs.getString(7);
    int NumOrdPresenti = rs.getInt(8);
    byte[]FlussoCZip = rs.getBytes(9);

    s = "IDFlussoC: " + String.valueOf(idFlussoC) +
    ", IDEnte: " + String.valueOf(idEnte) +
    ", StrutturaIPA: " + StrutturaIPA +
    ", Esercizio: " + Esercizio +
    ", NomeFile: " + NomeFile +
    ", Stato: " + Stato +
    ", DataCaricamento: " + DataCaricamento +
    ", NumeroOrdinativiPresenti: " + String.valueOf(NumOrdPresenti);
    ris.put(s,FlussoCZip);
   }
     break;

    case("OPI"):
        ...
    break;

    case("RicevuteServizio"):
    ...
    break;
      }
return ris;
```

```
}
  }
```

Come in precedenza alcune parti del codice superflue sono state tagliate e sostituite con i puntini di sospensione (...), questo sempre allo scopo di alleggerire la lettura e la comprensione dello stesso. La classe inizia con la dichiarazione delle variabili statiche "JDBC-DRIVER", DB-URL, USER e PASS. Esse saranno tutte inizializzate nel metodo costruttore con i parametri per la connessione al mio database. JDBC-DRIVER prenderà una stringa per il reperimento del driver JDBC necessario per la versione di SQL Anyware 17. A DB-URL sarà invece assegnato l'url a cui reperire il database e come prevedibile a USER e PASS il nome utente e la password per l'accesso alla base di dati. Per non tediare il lettore con l'eccessiva verbosità di tutti i metodi creati ad hoc, ne prenderemo solo uno in esame dato che essenzialmente, differiscono solo dalla query specifica che effettuano sul database e per i parametri che si estraggono dalla risposta dello stesso. Mi limiterò in seguito a specificare solo il compito di ognuno di essi in una breve lista. L'unico metodo che esamineremo è, come prevedibile, quello soprannominato "CercaOperatore" che però, verrà analizzato mettendo l'enfasi sulle fasi necessarie a dialogare con il database. Innanzitutto bisognerà dichiarare una variabile "Connection" che io ho, con molta fantasia, soprannominato "con"; ed una "Statement" detta "stmt". Ho poi dichiarato una stringa "Request" che verrà utilizzata come contenitore della query e un arry di stringhe per l'estrazione delle informazioni necessarie dalla risposta del database. Tutti i passaggi di comunicazione con la base di dati sono incapsulati in un try-catch con relative eccezioni specifiche nel caso quest'ultima non andasse a buon fine. Il primo passaggio da compiere è la registrazione del driver JDBC attraverso il comando "Class.forName(JDBC-DRIVER);". Si passa poi all'apertura della connessione con il DB assegnando alla variabile "conn" precedentemente dichiarata il comando "DriverManager.getConnection(DB-URL,USER,PASS);" che come prevedibile restituirà una connessione alla base di dati. L'ultima fase consiste nell'eseguire la query. Per farlo, bisognerà innanzitutto assegnare uno Statement della connessione creata prima alla variabile "stmt" precedentemente dichiarata attraverso il comando "conn.createStatement();". Il prossimo passo sarà assegnare alla stringa Request la query voluta e in fine eseguire la richiesta attraverso il comando "stmt.executeQuery(Requst);". Quest'ultimo restituirà un set di tipologia "ResultSet" denominato "rs" il quale, verrà usato come un iterator dato che, attraverso il comando "rs.next()", sarà possibile scandire ogni elemento del set di oggetti. Nel metodo, questi elementi saranno estratti ed assegnati all'array di stringhe dichiarato prima per essere poi restituito come valore di

ritorno del metodo. Lo stesso terminerà liberando l'ambiente attraverso i comandi "rs.close()", "stmt.close()" e "conn.close()". Passiamo ora alla lista dei metodi della classe con relativa spiegazione riguardante l'utilizzo:

- **CercaOperatore**: Metodo usato per controllare se un operatore di un ente è iscritto alla piattaforma da me sviluppata. Prende in input i dati dell'operatore che cerca di fare il login e se presente, restituisce dati relativi all'utente utili per l'esecuzione di altre domande al DB dopo di esso; altrimenti restituisce "null".

- **GET-IDENTITY**: Metodo usato dalle altre classi per richiedere alla base di dati la creazione di una nuova tupla di un certo tipo di dato come "FlussoCaricato" o "FlussoScaricato" in maniera da garantire la consistenza del DB stesso. Restituisce la chiave primaria della tupla creata.

- **INSERT**: Metodo che prende come parametro direttamente la stringa contenente la query da sottoporre al database. Esso è usato per svolgere le operazioni di inserimento nella base di dati, restituisce "ok" se l'operazione è andata a buon fine.

- **SELECTFS**: Metodo che effettua una ricerca relativa ai flussi scaricati presenti nel database in base a vari campi passati come parametro i quali possono anche essere nulli. Passa la risposta del DB ad un metodo privato che si occupa di formare l'HashMap che verrà restituita a chi utilizza il metodo.

- **SELECTFC**: Metodo che effettua una ricerca relativa ai flussi caricati presenti nel database in base a vari campi passati come parametro i quali possono anche essere nulli. Passa la risposta del DB ad un metodo privato che si occupa di formare l'HashMap che verrà restituita a chi utilizza il metodo.

- **SELECTOPI**: Metodo che effettua una ricerca relativa agli OPI presenti nel database in base a vari campi passati come parametro i quali possono anche essere nulli. Passa la risposta del DB ad un metodo privato che si occupa di formare l'HashMap che verrà restituita a chi utilizza il metodo.

- **formaRisultSELECT**: Metodo privato che ha il compito di riempire un'HashMap con i dati forniti dal DB ai tre metodi precedenti. Questa HashMap è così formata: innanzitutto c'è una riga per ogni tupla contenuta nella risposta del DB. Ogni riga della Map è formata da una

coppia (Chiave,Valore) dove; la chiave è una stringa contenente tutti i dati della tupla in questione e il valore è costituito da un array di byte contenente il file del flusso identificato dalla tupla.

I tre metodi SELECTFC, SELECTFS e SELECTOPI anche se molto simili non possono essere sostituiti da un unico metodo in quanto ognuno di essi svolge dei controlli con parametri di ricerca differenti su dati di entità diverse.

3.5.2 ReaderXML

Passiamo ora alla seconda classe ausiliaria denominata "ReaderXML". Come si vuol far comprendere dal nome della classe, il suo compito è quello agire su file XML nella forma di spacchettare e scomporre i flussi di cui ci occupiamo. Prima di esporre e commentare il codice conviene comprendere la struttura generale dei file XML che ho trattato. Nella pratica infatti, il flusso di OPI è costituito da un file di testo (con estensione.xml) composto da elementi e strutture (denominati tag in linguaggio XML), definiti secondo le regole previste dalla sintassi del linguaggio XML.

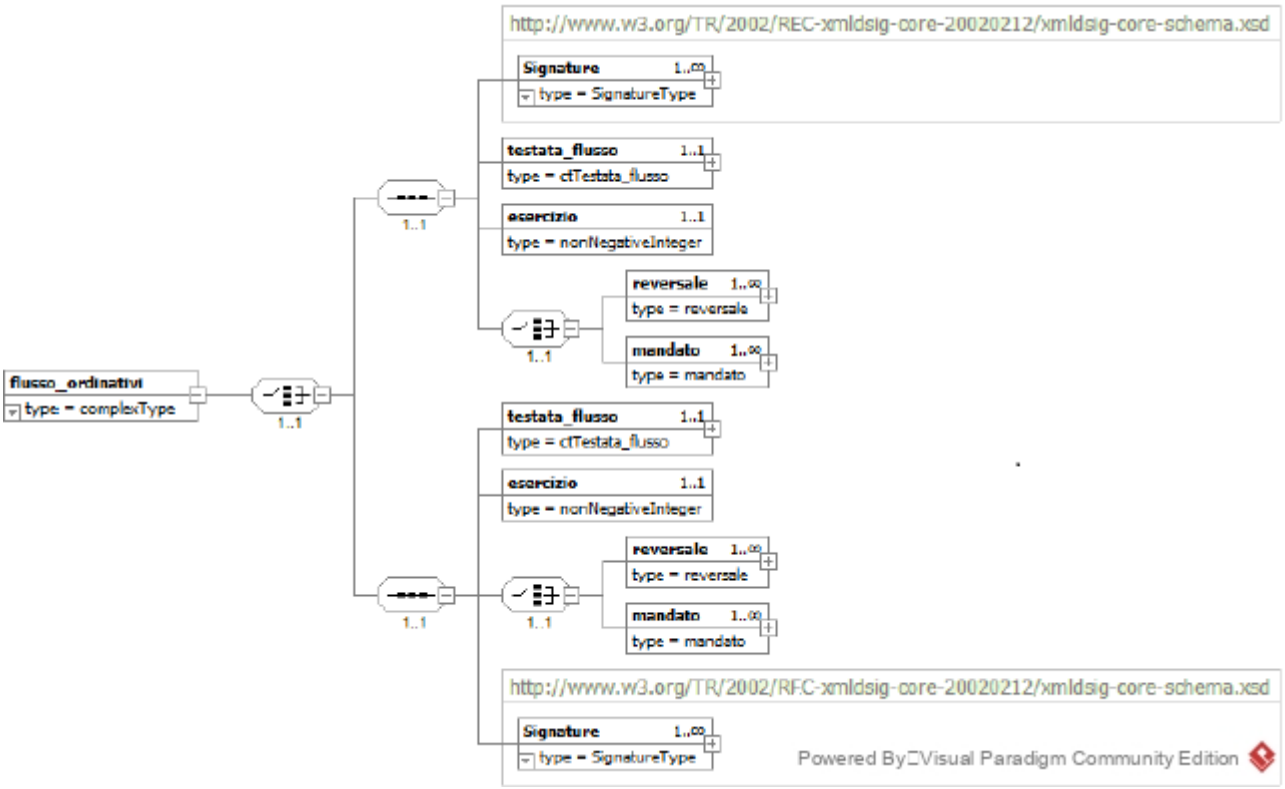

Figura7. Struttura del flusso OPI

La composizione di un flusso di ordinativi è riportata in Figura 7; da essa si evince che un flusso OPI è composto da una testata, dove sono riportate le informazioni per l'identificazione del flusso e per il suo indirizzamento, il dato che contiene l'anno di competenza di esercizio cui si riferisce il flusso e, di seguito, uno o più mandati / reversali. Oltre a queste informazioni, nel flusso è presente la struttura che contiene i dati relativi alla firma del flusso stesso (elemento <Signature>), struttura che può essere indicata come primo o come ultimo elemento del flusso stesso. Ora che abbiamo chiarito come sono strutturati i flussi sotto forma di .xml possiamo esporre il codice andando poi a commentarne il contenuto.

```
import java.io.*;
import java.util.*;
import org.apache.commons.codec.binary.Hex;
import org.jdom.*;
import org.jdom.input.*;
import org.jdom.output.*;
import OpWithDB.OpWithDB;

public class ReaderXML {

public ReaderXML() {}

public String[]  getDati(String nameFile){
   String[] s = new String[2];
   try {
        //Creo un SAXBuilder e con esco costruisco un document
        SAXBuilder builder = new SAXBuilder();
        File f = new File(nameFile);
        Document document = builder.build(f);

        //Prendo la radice
        Element root = document.getRootElement();

        //Estraggo i figli dalla radice
        List<?>children = root.getChildren();
        Iterator<?> iterator = children.iterator();

        //scorro la testata che e' sempre il primo figlio
```

```
        di questo tipo di file xml
        Element item = (Element)iterator.next();

        //arrivo al child esercizio
        item = (Element)iterator.next();
        s[0]=item.getText();

        //conto il numero di mandati/reversali
        int i = 0;
        while(iterator.hasNext()){
              item = (Element)iterator.next();
              i=i+1;
         }
         s[1] = String.valueOf(i);
    }
    catch (Exception e) {
         System.err.println("Errore durante la lettura dal file");
         e.printStackTrace();
    }
    return s;
}

public String IMPORTOPI(String fileName, int nOpi,
                        int IDFlussoC, int IDEnte,
                        String StrutturaIPA,String Esercizio)
                        throws JDOMException {
 for(int j = 0 ; j < nOpi ; j++) {
  try {

   //Creo un SAXBuilder e con esso costruisco un document
   SAXBuilder builder = new SAXBuilder();
   File f = new File(fileName);
   Document documentR = builder.build(f);

   //Creazione dell'oggetto XMLOutputter
   XMLOutputter outputter = new XMLOutputter();

   //Imposto il formato dell'outputter come "bel formato"
   outputter.setFormat(Format.getPrettyFormat());

   //inizializzo un fos
```

```
FileOutputStream FOS = new
FileOutputStream("versione_ordinativo_temp_"+j+".xml");

//produco l'output sul file scelto
outputter.output(documentR, FOS);

//chiudo il fos
FOS.flush();
FOS.close();

//creo un nuovo document che conterra' un unico opi
File OPI = new File("versione_ordinativo_temp_"+j+".xml");
Document documentOPI = builder.build(OPI);

//Prendo la radice
Element root = documentOPI.getRootElement();

//Estraggo i figli dalla radice
List<?>children = root.getChildren();
Iterator<?> iterator = children.iterator();
Element item=null;

//scorro la testata che e' sempre il primo figlio di
questo tipo di file xml
item = (Element)iterator.next();

//scorro il child esercizio
item = (Element)iterator.next();

//aggiungo alla listaDaRimuovere tutti i child mandato o
reversale tranne il child j
int i = 0;
HashMap<Integer,Element> listaDaRimuovere = new HashMap<Integer,Element>();

while(iterator.hasNext()){
 item = (Element)iterator.next();
 if(j != i) {listaDaRimuovere.put(i,item);}
 i=i+1;
}

//rimuovo tutti i child della listaDaRimuovere
```

```
for(i=0; i <= listaDaRimuovere.size(); i++) {
 if(j != i) {root.removeContent(listaDaRimuovere.get(i));}
}
String  TipoOrdinativo, TipoOperazione, DataOrdinativo ;
float Importo;
byte[] Filexml;
List<?>children1 = root.getChildren();
Iterator<?> iterator1 = children1.iterator();

//scorro la testata che e' sempre il primo figlio di
questo tipo di file xml
item = (Element)iterator1.next();

//scorro il child esercizio
item = (Element)iterator1.next();

//arrivo al mandato o al reversale
item = (Element)iterator1.next();

//setto il tipo di opi
TipoOrdinativo = item.getName();

//setto il tipo dell'ordinativo
TipoOperazione = item.getChildText("tipo_operazione");

//setto la data dell'ordinativo
DataOrdinativo =
item.getChildText("data_"+TipoOrdinativo);

//setto l'importo dell'opi
Importo=Float.parseFloat(item.getChildText
                        ("importo_"+TipoOrdinativo));

//inizializzo un oggetto di tipo OpWithDB
OpWithDB op  = new OpWithDB();

//creo un nuovo fos
FileOutputStream FOS1 = new
FileOutputStream("versione_ordinativo_temp_"+j+".xml");

//modofico il file aggiornandolo al nuovo documentOPI
```

```
   outputter.output(documentOPI, FOS1);

   //chiudo il fos
   FOS1.flush();
   FOS1.close();

   //salvo il file in formato byte[] e lo elimino
   Filexml = fileToByte("versione_ordinativo_temp_"+j+".xml");

   //iserisco l'opi nel database
   op.INSERT(...);

 }
 catch (IOException e) {...}

 return "ok";
 }

 private byte[] fileToByte(String namefile) throws IOException {...}

}
```

Come la classe precedente anche questa fà da contenitore per dei metodi creati ad hoc per svolgere alcuni compiti utili alla classe principale "ClientApplication". Questi metodi si concentrano sul manipolare file XML e sono essenzialmente due tralasciando un metodo privato che trasforma un file in un array di byte e ovviamente il metodo costruttore.

- **getDati**: questo metodo prende come parametro il nome di un file .xml della tipologia sopra esplicata e restituisce un coppia di stringhe contenenti il numero dei mandati/reversali del flusso in questione e l'anno di competenza dell'esercizio a cui il flusso è relativo. I suddetti dati sono necessari per il salvataggio dei flussi caricati dalla piattaforma su SIOPE, all'interno del DB. Nel metodo viene innanzi tutto creato un elemento di tipologia "Document" per il trattamento del file in questione come un albero, il quale avrà una radice e dei figli che nel nostro caso sono costituiti dalla testata, dall'esercizio e dai vari mandati-reversali. Il codice prosegue estraendo la radice dall'oggetto di tipo "Document" attraverso il comando "document.getRootElement()" che restituirà un oggetto di tipologia "Element" che io ho nominato "root". In seguito

da questo elemento "root" si estrae la lista dei figli attraverso il comando "root.getChildren()". Su di questa lista ho poi creato un iterator da scorrere. Grazie ad esso ho estratto il secondo figlio della radice e cioè il campo "esercizio" per poi salvarlo nella prima posizione della coppia di stringhe da restituire. Ho poi sfruttato l'itarator per scorrere tutti gli altri elementi della lista di figli della radice che sapevo essere tutti mandati o reversali per contarli e quindi salvarne il numero nel secondo campo della coppia da mandare al chiamante del metodo.

- **IMPORTOPI**: Data la grande quantità di piccoli passaggi presenti in questo metodo, che potrebbero far perdere la visione generale al lettore, ho deciso di tralasciarne molti in favore di una più chiara comprensione dei meccanismi principali in esso presenti. Partiamo dicendo che il metodo, preso un flusso sotto forma di XML contenente tanti mandati-reversali, avrà il compito di dividerlo in tanti piccoli OPI (flussi) ognuno dei quali conterrà un solo mandato-reversale per poi salvarli uno per uno nel DB. Il metodo prende come due dei parametri iniziali il numero di mandati-reversali e il nome del file .xml. Metterà quindi in piedi un ciclo che scorrerà il numero di mandati-reversali con un contatore "j". In esso, si crea ogni volta una copia del file .xml passato come parametro per poi andare a cancellarne tutti i figli di tipologia mandati-reversali tranne il j-esimo così da avere ogni volta un OPI con testata e esercizio uguale al flusso principale ed un solo mandato-reversale sempre diverso. Il passaggio successivo, sempre all'interno del ciclo, consiste nell'estrarre da ognuno di questi OPI tutti i dati a loro necessari per il salvataggio sul DB. Questo viene fatto considerando il nodo mandato-reversale come root e scorrendone i figli al cui interno si trovano tutti i dati a noi utili. Per l'inserimento nel database si utilizza il metodo "INSERT" della classe "OpWithDB".

3.5.3 Ispector RicevuteApplicatio, RicevuteServizio e GiornaleCassa

Queste tre classi saranno trattate contemporaneamente visto che il loro funzionamento è il medesimo. L'unica cosa che le differenzia sono i dati che manipolano; il quale è un aspetto trascurabile ai fini della corretta comprensione delle attività che svolgono. Ne prenderemo, quindi, in esame una sola delle tre per commentarla.

```
import java.io.IOException;
import java.nio.charset.Charset;
```

```
import java.util.Arrays;
import java.util.Iterator;
import org.apache.commons.codec.binary.Hex;
import org.springframework.http.HttpEntity;
import org.springframework.http.HttpHeaders;
import org.springframework.http.HttpMethod;
import org.springframework.http.MediaType;
import org.springframework.http.ResponseEntity;
import org.springframework.web.client.RestClientException;
import org.springframework.web.client.RestTemplate;
import ListaEsitiFlussoOrdinativi.*;
import OpWithDB.OpWithDB;


public class IspectorRicevuteServizio implements Runnable {
       private String[] s;
       public IspectorRicevuteServizio(String[] s) {
              this.s=s;
       }

       public void run() {
           RestTemplate restTemplate = new RestTemplate();
           OpWithDB op = new OpWithDB();
           ResponseEntity<ListaEsitiFlussoOrdinativi> response;

           //scarico la lista delle ricevute di servizio relative al mio ente
           non ancora scaricate
           try {

                response = restTemplate.exchange(
                "http://localhost:8080/...",
                HttpMethod.GET,setHeaderGETList(),
                ListaEsitiFlussoOrdinativi.class);
                ListaEsitiFlussoOrdinativi list = response.getBody();
                Iterator<infoEsiti> flusso = list.getL().iterator();
                int j = 2; // per prova

                //per ogni elemento della lista...
                while(j != 0 /*flusso.hasNext()*/) {
                     infoEsiti info  = flusso.next();
                     int IDFlussoS = op.GET_IDENTITY("FlussiScaricati");
```

```
                    //insersco una riga nella tabella FlussiScaricati
                    op.INSERT(...);
                    //scarico la ricevuta di servizio
                    ResponseEntity<byte[]> response1 = restTemplate.exchange(
                    "http://localhost:8080/...",
                    HttpMethod.GET,setHeaderGETDown(), byte[].class);

                    //inserisco una riga nella tabella RicevuteServizio
                    op.INSERT(...);
                    j--;
             }
          } catch (RestClientException | IOException e) {e.printStackTrace();}
       }

       public static HttpEntity<String> setHeaderGETDown() throws IOException{
              HttpHeaders headersdf = new HttpHeaders();
              headersdf.setAccept(Arrays.asList(new MediaType("application",
              "zip")));
              headersdf.set("Host","<siope+>");
              return new HttpEntity<String>(headersdf);
       }


       public static HttpEntity<String> setHeaderGETList()throws IOException{
              HttpHeaders headersdf = new HttpHeaders();
              headersdf.setAccept(Arrays.asList(
              new MediaType(MediaType.APPLICATION_JSON, Charset.defaultCharset()
              headersdf.set("Host","<siope+>");
              return new HttpEntity<String>(headersdf);
       }
}
```

Il commento inizia svelando al lettore che la famiglia di classi di tipologia "Ispector", come facilmente intuibile dall'implementazione dell'interfaccia "Runnable", istanzia una serie di task possibilmente svolgibili da dei thread. Essi infatti, implementano il metodo "run()" al cui interno è presente il compito che il thread si ritroverà a svolgere, più due metodi ausiliari ad esso per la formazione del header delle richieste http per contattare i servizi forniti dal progetto denominato "SIOPE" di tipologia GET. Dato che ho esposto il codice della classe "IspectorRicevuteServizio", prenderemo come dati trat-

tati quelli di tipologia "EsitiFlussoOrdinativo". Senza entrare anche questa volta nello specifico e ponendo l'enfasi solo sul funzionamento generale del metodo; diciamo che esso innanzi tutto si occupa di reperire la lista degli esiti dei flussi ordinativi non ancora scaricati dalla piattaforma; andando a contattando il servizio ListaEsitiFlussoOrdinativi di tipologia GET fornito dallo stud "SIOPE" settando il campo "download" dell'URL della richiesta http uguale a false. Ho poi ricavato un iterator della lista per scorrerne ogni elemento all'interno di un ciclo while allo scopo di caricare per ogni elemento della lista una tupla sulla tabella del DB denominata "FlussiScaricati". Sempre all'interno del while, quindi per ogni elemento della lista scaricata, ho contattato il servizio "DownloadEsitoFlussoOrdinativi" di tipologia GET, anch'esso fornito dal progetto "SIOPE", per scaricare il file zip contenente l'EsitoFlusso in formato byte array. Avendo ora tutti i dati necessari per il corretto inserimento di una riga nella tabella "RicevuteServizio" del DB, faccio proprio questo sfruttando il metodo "INSERT" fornito dalla classe "OpWithDB".

3.5.4 ClientApplication

Terminiamo la carrellata di classi del progetto "Client" con la classe "ClientApplication" la quale usa tutte le altre allo scopo di creare uno scheletro per un eventuale interfaccia. In essa infatti sono raggruppate tutte le funzionalità di cui la piattaforma potrà usufruire contattando lo stub "SIOPE" e interagendo con la base di dati preposta al salvataggio dei dati.

```
import...

@SpringBootApplication
public class ClientApplication {


public static void main(String[] args) throws IOException,
                                        NumberFormatException, JDOMException {

SpringApplication.run(ClientApplication.class, args);
OpWithDB Op = new OpWithDB();
String operazione = "login";
```

```
boolean passa = false;
String[] nc = null ;

while (passa == false) {
  switch(operazione){
    case "login":
        if(( nc = Op.CercaOperatore("1175","2","'arehtbdrt'"))  == null )
              System.out.println("Registrati");
        else {
              passa = true;
              System.out.println("benvenuto "+ nc[0] + " "+ nc[1]);
        }
   break;

  case "register" :
    String s = Op.INSERT(...);
    if(s == "ok")
    System.out.println("Registrazione effettuata con successo");
  break;
 }
}

boolean chiudi = false;
RestTemplate restTemplate = new RestTemplate();



/*while(chiudi == false) {*/
//parametri per la ricerca nel DB
String Prog =null;
String DataDa = null;
String DataA = null;
String nomefile = null;
String esercizio = null;
String TipoOperazione = null;
String TipoOrdinativo = null;
String NumOrdinativoDa = null;
String NumOrdinativoA = null;
HashMap<String,byte[]> ris = null;

IspectorRicevuteServizio ispectorRS=new IspectorRicevuteServizio(nc);
```

```
Thread threadRS = new Thread(ispectorRS);
threadRS.start();

IspectorRicevuteApplicativo ispectorRA=new IspectorRicevuteApplicativo(nc);
Thread threadRA = new Thread(ispectorRA);
threadRA.start();

IspectorGiornaleCassa ispectorGC = new IspectorGiornaleCassa(nc);
Thread threadGC = new Thread(ispectorGC);
threadGC.start();

operazione = "ImportaFlusso";
switch(operazione ) {

  case("ImportaFlusso"):
      String NomeFile = "flusso_opi_importato_18517502.xml";
      ReaderXML r = new ReaderXML();
      String[] dati = r.getDati(NomeFile);
      HttpEntity<byte[]> up =
      setHeaderAndBodyPOST("flusso_opi_importato_18517502.zip");

      ResponseEntity<UploadFlussoOrdinativi> response =
      restTemplate.exchange("http://localhost:8080/...,
                            HttpMethod.POST,up,UploadFlussoOrdinativi.class);

      int IDFlussoC = Op.GET_IDENTITY("FlussiCaricati");

      Op.INSERT(...);

      System.out.println(r.IMPORTOPI(NomeFile, Integer.parseInt(dati[1]),
                         IDFlussoC,Integer.parseInt(nc[2]), nc[3], dati[0]));

   break;

   case("ListaFlussiImportati"):
       if(nomefile != null) { nomefile = " NomeFile= " + nomefile;}
       if(DataDa != null) {DataDa = "DataCaricamento >= " + DataDa;}
       if(DataA != null)  {DataA =  "DataCaricamento <= " + DataA;}
       if(esercizio != null) {esercizio = "esercizio=" + esercizio;}

       ris = Op.SELECTFC("FlussiCaricati",nomefile,DataDa,DataA,esercizio);
```

```
        if(ris!= null) {
           Iterator<byte[]> iterV = ris.values().iterator();
           Iterator<String> iterK = ris.keySet().iterator();
           int i = 0;
           while(iterK.hasNext()) {
                 String row =(String)iterK.next();
                 byte[] bytes =(byte[])iterV.next();
                 System.out.println(row);
                 createDownloadedZipFile(bytes,"FlussoCaricoato"+i+".zip");
                 i=i+1;
           }
        }
   break;

   case("ListaOPIEnte"):
   ...
   break;

   case("RicevuteServizio"):
   ...
   break;

   case("GiornaliDiCassa"):
   ...
   break;

//funzioni ausiliarie
public static HttpEntity<byte[]> setHeaderAndBodyPOST(String namefile)
                                 throws IOException{
       //set header
       HttpHeaders headers = new HttpHeaders();

       headers.setAccept(Arrays.asList(new MediaType(MediaType.APPLICATION_JSON,
                                                     Charset.defaultCharset())))
       headers.setContentType(new MediaType("application", "zip"));
       headers.setContentLength(10000000);
       headers.set("Host","<siope+>");

       //set body
       FileInputStream is = new FileInputStream(namefile);
       ZipInputStream zis = new ZipInputStream(is);
```

```
        ByteArrayOutputStream byteArrayOutputStream =
                                new ByteArrayOutputStream();
        BufferedOutputStream bufferedOutputStream =
                                new BufferedOutputStream(byteArrayOutputStream);
        ZipOutputStream zipOutputStream =
                                new ZipOutputStream(bufferedOutputStream);

        ZipEntry ingresso;
        while ( ( ingresso = zis.getNextEntry()) != null  ) {
             zipOutputStream.putNextEntry(new ZipEntry(ingresso.getName()));
             FileInputStream fileInputStream =
                                    new FileInputStream(ingresso.getName());
             IOUtils.copy(fileInputStream, zipOutputStream);
             fileInputStream.close();
             zipOutputStream.closeEntry();
        }

       if (zipOutputStream != null) {
             zipOutputStream.finish();
             zipOutputStream.flush();
             IOUtils.closeQuietly(zipOutputStream);
       }
       IOUtils.closeQuietly(bufferedOutputStream);
       IOUtils.closeQuietly(byteArrayOutputStream);
       is.close();
       zis.close();
       return new HttpEntity<byte[]>
                  (byteArrayOutputStream.toByteArray(),headers);
}

public static void createDownloadedZipFile(byte[] bytes,String fileName)
                                                        throws IOException {
         File f = new File(fileName);
         f.createNewFile();
         FileOutputStream fis = new FileOutputStream(f);
         fis.write(bytes);
         fis.flush();
         fis.close();
}

public static  String getDate() {
```

```
        Calendar calendar = new GregorianCalendar();
        int ore = calendar.get(Calendar.HOUR)+1;
        int minuti = calendar.get(Calendar.MINUTE)+1;
        int secondi = calendar.get(Calendar.SECOND);
        int millis = calendar.get(Calendar.MILLISECOND);
        int giorno = calendar.get(Calendar.DAY_OF_MONTH);
        int mese = calendar.get(Calendar.MONTH)+1;
        int anno = calendar.get(Calendar.YEAR);
        return anno + "-" + mese + "-" + giorno + " " + ore +
               ":" + minuti + ":" + secondi + "." + millis;
}

}
```

La classe è in pratica, quella creata dal servizio Spring Boot initializr per il progetto Maven "Client". Contiene quindi il metodo main che lancia l'intera applicazione web risultante. Seguendo sempre la scia dei commenti precedenti ci focalizzeremo solo sulle parti salienti del codice. All'interno del main la prima cosa fatta è un controllo riguardante l'identità dell'utente che cerca di utilizzare la piattaforma. Infatti se tenta di fare il login l'applicazione controllerà nel DB se l'utente è presente attraverso il metodo "CercaOperatore", della classe "OpWithDB". Se esso è presente il login viene effettuato con successo e il metodo ci restituisce dei dati utente che ci saranno utili in seguito altrimenti consiglia all'utente di registrarsi. La registrazione è implementata di seguito sfruttando una chiamata al metodo "INSERT" sempre della classe "OpWithDB". Una volta che l'utente avrà effettuato il login con successo verranno attivati tre thread che si occuperanno rispettivamente dei task contenuti nelle classi "IspectorRicevuteServizio", "IspectorRicevuteApplicativo" e "IspectorGiornaleCassa" dato che il compito di tenere aggiornato il DB con i nuovi flussi presenti su "SIOPE" non è propriamente dell'utente bensì della piattaforma stessa. Di seguito è presente uno switch con cui l'utente potrà scegliere quale funzionalità usare tra il:

- caricare nuovi flussi su "SIOPE" e di conseguenza anche sul DB;
- reperire la lista di flussi importati dal DB;
- reperire la lista degli OPI dell'ente di cui fà parte l'utente sempre dalla base di dati;
- reperire la lista delle ricevute di servizio, anch'essa dal database;
- reperire la lista dei giornali di cassa dell'ente dal DB.

Di queste cinque andremo a commentare solo la prima e la seconda dato che per implementazione le altre sono identiche alla seconda salvo che per i dati manipolati ed i controlli effettuati. La funzionalità di importazione flusso è il primo caso dello switch e si differenzia dalle altre dato che, invece di reperire dati, ha il compito di caricarne su "SIOPE" andando a contattare l'unico servizio di tipo POST fornito da esso. Dopo di che, salva il flusso appena caricato sul DB, per poi, attraverso il metodo "IMPORTOPI" della classe ausiliaria "ReaderXML", dividere il flusso in tanti piccoli OPI uno per mandato-reversale e caricare anch'essi sul database. La funzionalità di stampa della lista dei flussi importati secondo parametri di ricerca, invece, andrà prima di tutto ad effettuare delle manipolazioni sulle stringhe contenenti i parametri in questione , per poi contattare il metodo "SELECTFC" della classe ausiliaria "OpWithDB" a cui passerà prorpio questi parametri per la selezione della lista da reperire. La funzionalità termina con un while che scorre due iterator, creati sulla hash map risultate dal metodo prima contattato, allo scopo di stamparne le righe contenenti i dati che richiedevamo. Oltre al metodo main la classe contiene dei metodi ausiliari per il settaggio del contenuto della richiesta di tipologia POST, per la creazione di un file di tipo zip a partire da un file normale e per la costruzione di una data da passare come stringa che non tratteremo data l'eccessiva verbosità che li contraddistingue.

3.6 Creazione della base di dati

Dopo aver terminato la fase del tirocinio relativa alla creazione del modulo "Client" e dello stub "SIOPE"; argomento principale su cui avrebbe dovuto vertere il mio lavoro di tirocinio in accordo con il tutor aziendale ho deciso di ampliare l'elaborato andando ad implementare anche un database per il salvataggio dai dati trattati dalla piattaforma. Come detto nella parte relativa alle tecnologie utilizzate il DB è stato realizzato con l'applicazione "SQL Anyware 17 developer" e in pratica non ho dovuto fare altro che traslare sotto forma di tabelle il modello ER precedentemente realizzato. Dopo la creazione del database all'interno dell'applicazione, sempre grazie alle sue funzionalità ho attivato un servizio che mi è servito per esporre il DB appena creato su di un url che è proprio quello a cui si connetteva la classe OpWithDB". Questo servizio per essere creato ha necessitato dell'ausilio del connettore ODBC esposto nel capitolo relativo alle tecnologie utilizzate.

Capitolo 4

Conclusioni

Durante il periodo di tirocinio, ho implementato una piattaforma di interfacciamento per il sistema SIOPE+. Essa funge da intermediario tra gli Enti dell'amministrazione pubblica e il sistema SIOPE+. Questa piattaforma è formata da un modulo Client che si occuperà di comunicare con SIOPE+ ed un database su cui verranno salvati tutti i flussi che passano per essa e tutti i dati utili al suo funzionamento. Inoltre, per testare la piattaforma ho implementato anche uno stub di SIOPE+ che andasse a mimarne il funzionamento.

Lo sviluppo di questa piattaforma ha comportato non solo l'applicazione delle mie conoscenze universitarie, ma anche uno studio approfondito delle tecnologie a me non familiari necessarie per svilupparla. Inoltre ha richiesto un'attenta analisi del problema in questione per poter organizzare al meglio i dati utilizzati, evitare incongruenze e fare in modo che il sistema sia sempre consistente.

L'applicazione sviluppata, attualmente, non è utilizzata in quanto, pur avendo testato singolarmente le parti, essa necessita ancora di alcuni controlli per assicurare che il software rispetti i requisiti e le specifiche richieste. Gli sviluppatori dell'azienda dovranno individuare le carenze di correttezza, completezza e affidabilità delle componenti software da me sviluppate in modo tale da correggerne gli errori e produrre, così, la versione successiva. Le tecnologie e gli strumenti utilizzati sono stati adeguati alle esigenze del progetto, infatti hanno semplificato molto l'implementazione e sono state non molto difficili da padroneggiare.

4.1 Competenze acquisite

Grazie allo svolgimento di questo progetto ho raggiunto dei risultati dal punto di vista didattico che corrispondono con gli obiettivi prefissati dall'azienda. In particolare ho:

- approfondito le conoscenze del linguaggio di programmazione Java;
- imparato ad utilizzare alcune funzionalità dell'IDE Eclipse di cui non ero a conoscenza;
- appreso la struttura e l'utilizzo di file XML;
- analizzato le specifiche ed i requisiti per la realizzazione del progetto;
- imparato ad utilizzare il framework Spring Boot;
- imparato ad utilizzare Apache Maven per la gestione dei progetti;
- realizzato dei web services REST;
- utilizzato un database relazionale per l'archiviazione dei dati.

4.2 Pensieri conclusivi generali sull'esperienza

Questa esperienza mi ha dato l'opportunità di capire cosa si prova a lavorare al fianco di altre persone che ti offrono spunti ed idee. Personalmente non essendomi mai trovato in una situazione simile non sapevo quanto studio e lavoro preliminare ci fosse prima dell'implementazione vera e propria di un software. Infatti, la parte che mi è risultata più difficile è stata proprio quella relativa alla comprensione di cosa sarei dovuto andare a realizzare. Inoltre mi sarebbe piaciuto ampliare ulteriormente il progetto andando a sviluppare anche l'interfaccia utente. Infine posso dire che l'esperienza nel complesso è risultata positiva.

Bibliografia

[1] *EASYSIOPE, Applicazione web per gli utenti di SIOPE+*. URL: https://siopeplus.vaservices.eu:444/easysiopeente/#/.

[2] *Eclipse, Ambiente di sviluppo*. URL: https://www.eclipse.org.

[3] *java.io, Libreria java*. URL: https://docs.oracle.com/javase/7/docs/api/java/io/package-summary.html.

[4] *java.nio, Libreria java*. URL: https://docs.oracle.com/javase/7/docs/api/java/nio/package-summary.html.

[5] *java.sql, Libreria java*. URL: https://docs.oracle.com/javase/7/docs/api/java/sql/package-summary.html.

[6] *java.util, Libreria java*. URL: https://docs.oracle.com/javase/7/docs/api/java/util/package-summary.html.

[7] *Maven, Framework*. URL: https://maven.apache.org/guides/.

[8] *ODBC, Tecnologia*. URL: https://docs.microsoft.com/it-it/sql/odbc/microsoft-open-database-connectivity-odbc?view=sql-server-ver15.

[9] *Postman, Applicazione*. URL: https://www.postman.com/.

[10] *SIOPE+ Regole di Colloquio*. URL: http://www.rgs.mef.gov.it/_Documenti/VERSIONE-I/e-GOVERNME1/SIOPE/SIOPE/Regole_di_Colloquio.pdf.

[11] *Sito governativo per la piattaforma SIOPE*. URL: https://www.agid.gov.it/it/piattaforme/siope/standard-opi-gruppo-lavoro..

[12] *Spring Boot, Framework*. URL: https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/.

[13] *Spring Initializr*. URL: https://start.spring.io/.

[14] *SQL Anyware 17, Applicazione*. URL: https://www.sap.com/products/sql-anywhere.html.