**DISPATCHER UNIFICATO – MANUALE USO**

**SCOPO**

Servizi per l’invio di messaggi al backend di AppIO.

Il presente documento ha lo scopo di descrivere il funzionamento di 3 componenti distinte:

1. **Invio sincrono dei messaggi**

Viene effettuato invocando direttamente la componente WSO2 (modulo **ClientDispatcher**)

1. **Riempimento della coda di invio**

Il servizio permette di inserire righe nella tabella IO\_CODA con opportuni dati relativi al tipo di invio e al giorno ed ora prescelti (modulo **LoaderMessageQueue**)

1. **Invio asincrono dei messaggi**

La coda riempita al punto 2 viene processata da uno scheduler java che invoca la stessa componente WSO2 del punto 1 per inviare i messaggi (modulo **SenderScheduler**)

Il risultato degli invii, sia sincroni che asincroni, viene registrato nella tabella di esiti IO\_ESITI da WSO2.

Vedere il diagramma .jpg allegato per la struttura completa del sistema.

**SEZIONE 1 – ELEMENTI COMUNI**

Tabelle coinvolte:

1. **IO\_CODA**: coda per invii asincroni. La tabella contiene per ogni record un messaggio pronto da inviare
2. **IO\_CONFIGURAZIONE**: elenco dei servizi configurati. Solo gli invii con TIPO\_BATCH presenti possono essere effettuati. Se si vuole aggiungere un nuovo servizio, occorre aggiungere una riga qui (vedi Gestione dei servizi)
3. **IO\_ESITI**: elenco degli esiti degli invii
4. **IO\_INTERVALLO\_INVIO**: schema per suddividere gli invii in fasce orarie (vedi Gestione dei servizi)

**SEZIONE 2 – INVIO SINCRONO**

Modulo: **ClientDispatcher**

Per inviare un messaggio sincrono è sufficiente invocare il servizio POST: **/dispatcherUnificato/notifica** da un client esterno (postman o batch). Lo swagger allegato **AppIO\_Client\_Dispatcher.yaml** contiene tutti i dettagli della chiamata. Ad ogni chiamata corrisponde un solo messaggio con il suo esito, in quanto l’invio di messaggi AppIO è in relazione 1:1 con il CF del cittadino. Il risultato dell’invio viene riportato nella tabella IO\_ESITI.

Spiegazione dettagliata dei campi del body:

1. fiscal\_code: CF del cittadino destinatario
2. token: chiave primaria del servizio AppIO
3. time\_to\_live: tempo massimo di tentativi di invio in secondi (default 3600)
4. tipo\_invio: nome del servizio da invocare. Deve essere preventivamente registrato sul DB seguendo gli step del paragrafo Gestione dei servizi
5. id\_coda: eventuale id se si effettua un invio massivo scodando una coda aggiuntiva
6. note: eventuali note esplicative che verranno salvate negli esiti
7. subject: subject del messaggio AppIO
8. markdown: testo del messaggio AppIO in formato markdown
9. due\_date: eventuale data di reminder per il calendario di AppIO
10. amount: eventuale pagamento in euro cents
11. notice\_number: iuv del pagamento
12. invalid\_after\_due\_date: indica se invalidare la possibilità di pagamento trascorsa la due\_date o meno (default false)

**GESTIONE DEI SERVIZI**

Per aggiungere un nuovo servizio occorre inserire una riga nella tabella **IO\_CONFIGURAZIONE**. Solo gli invii con **TIPO\_BATCH** presenti possono essere effettuati.

Spiegazione dettagliata delle colonne della tabella:

1. tipo\_batch: nome del nuovo servizio
2. token: chiave primaria del servizio AppIO
3. payload\_type: STATICO se il messaggio è fisso, DINAMICO se il messaggio cambia per ogni cittadino
4. subject: subject del messaggio AppIO
5. time\_to\_live: tempo massimo di tentativi di invio in secondi (default 3600)
6. ripetibile: MAI se l’invio deve essere tentato una volta sola, ERRORE se l’invio deve essere ritentato nei soli casi di errori del backend AppIO (429 e 500), SEMPRE l’invio viene rieseguito anche se il record è già presente tra gli esiti, indipendentemente dal codice della risposta di tale esito
7. messaggio\_base: coincide con il messaggio finale nel caso di invio STATICO, oppure viene utilizzato come messaggio base da elaborare nel caso di invio DINAMICO. Per andare a capo usare il simbolo “QQ”. Tale simbolo verrà poi tradotto in “\n” dal batch al momento dell’invio.

Esempio: Questo è un messaggio di prova. QQ Questa è una nuova riga. QQ QQ Questa è una nuova riga dopo una riga vuota

Si traduce in:

Questo è un messaggio di prova.

Questa è una nuova riga.

Questa è una nuova riga dopo una riga vuota

1. note: eventuali note esplicative che verranno salvate negli esiti
2. payload\_mex: non più utilizzata
3. id\_stato\_rec: non più utilizzata
4. tipo\_desc\_estesa: stringa utilizzata per visualizzare il servizio sul cruscotto elastic

**SEZIONE 3 – RIEMPIMENTO DELLA CODA DI INVIO**

Modulo: **LoaderMessageQueue**

**SCHEDULER JAVA PER CARICAMENTO DATI**

Per semplificare l’invio asincrono e/o massivo di messaggi è stata creata la componente Java – Spring Boot contenuta nel modulo **LoaderMessageQueue,** che permette di aggiungere record nella tabella di coda. Il caricamento avviene tramite la schedulazione di un opportuno timer quartz, quindi può essere effettuato in qualsiasi momento e configurato in anticipo. I dettagli dei servizi sono contenuti nello swagger AppIO\_Loader\_Message\_Queue.yaml

Gli endpoint di uso generale sono i seguenti:

GET: [appio/loader/timer](http://192.168.154.109:8080/LD_Loader_ComGE/comge/loader/timer)

Per avere la lista di tutti i timer attivi. Tale lista viene anche estratta ogni 15 minuti da un processo demone che la registra su file nella cartella /conf del tomcat.

DELETE: appio/loader/timer?id=”timer\_name”

Per cancellare un timer chiamato “timer\_name”.

Un discorso a parte va dedicato alla creazione di un nuovo timer di riempimento della coda. Infatti, visto che tale operazione dipende esplicitamente dal tipo di messaggio che si vuole inviare e dalla platea dei fruitori, non è possibile creare un’unica interfaccia, ma occorre creare job specifici. All’interno del modulo **LoaderMessageQueue** è stato lasciato un esempio, contenuto nella classe /src/main/java/it/appio/loader/timer/job/Job\_EDU\_BOLLETTINO\_RISTO.java e associato alla risorsa appio/loader/scolastico/eduBollettinoRisto

Se si vogliono creare dei nuovi timer, è sufficiente seguire questa struttura apportando poi le modifiche necessarie.

**FILE DI PROPERTIES**

Come tutti i progetti Spring Boot, il loader ha bisogno di un file di configurazione da creare sotto la cartella /conf del tomcat con il nome: LoaderMessageQueue.properties. Tale file deve contenere i dettagli dei DB utilizzati e le info necessarie ai vari timer. Nella cartella Properties del progetto viene riportato un esempio.

**ESEMPIO CREAZIONE NUOVO TIMER**

Per creare un nuovo timer di caricamento della coda si possono seguire i passi seguenti:

1. Creare il nuovo servizio nella tabella IO\_CONFIGURAZIONE come spiegato allo step “Gestione dei servizi”
2. Editare il file di configurazione LoaderMessageQueue.properties con i dati necessari all’invio, seguendo il modello:
   1. loader.nuovo.servizio.ambiente.target = PROD (oppure TEST se si vuole fare una prova sul DB di test)
   2. loader.nuovo.servizio.tipo.messaggio = nuovo tipo\_batch creato allo step 1
   3. loader.nuovo.servizio.data.invio.base = giorno inizio invii nel formato YYYY-MM-DD
   4. loader.nuovo.servizio.invii.al.giorno = numero di invii giornalieri (oppure 0 se l’invio è unico)
   5. loader.nuovo.servizio.orario = orario di invio usando il valore di ID\_INTERVALLO\_INVIO come appare nella tabella IO\_INTERVALLO\_INVIO
   6. loader.nuovo.servizio.data.scadenza = giorno limite oltre il quale non procedere più con gli invii nel formato YYYY-MM-DD
   7. loader.nuovo.servizio.note = lasciare vuoto oppure inserire eventuali note
   8. loader.nuovo.servizio.promemoria = lasciare vuoto oppure inserire la data promemoria AppIO nel formato string-date time (YYYY-MM-DDTHH:mm:ss)
   9. loader. nuovo.servizio.top = se TRUE limita la query sul DB dei dati sorgenti alle prime 5 righe
3. Estrarre i timer attualmente attivi con la chiamata GET spiegata in precedenza
4. Creare la struttura java necessaria:
   1. Nuova risorsa nella classe it.appio.loader.controller.LoaderController
   2. Nuovi metodi nelle classi di service appropriate it.appio.loader.service
   3. Aggiunta degli oggetti Entity e Repository per mappare i dati sorgenti
   4. Nuovo job sotto it.appio.loader.timer.job
5. Fare redeploy del war
6. Schedulare il nuovo timer: appio/loader/”nuovo\_servizio”

con body:

{

"cron\_schedule": "schedulazione in formato cron",

"initial\_offset\_ms": 5000,

"timer\_name": "Nome\_nuovo\_timer"

}

Per il formato cron vedere il documento ufficiale: <http://www.quartz-scheduler.org/documentation/quartz-2.3.0/tutorials/crontrigger.html>

1. Ricreare gli altri timer esistenti letti con la GET precedente (tranne demone che si crea da solo)
2. I log del processo sono nella cartella /logs del tomcat nel file specifico: LoaderMessageQueue.log

**SEZIONE 4 – INVIO ASINCRONO**

Modulo: **SenderScheduler**

L’invio asincrono viene eseguito da un batch java gestito a sua volta da uno scheduler. Il codice è contentuo nel modulo **SenderScheduler**. Ad ogni trigger del timer, il batch procede all’invio di tutti i messaggi presenti nella tabella di coda che sono schedulati per quel giorno e quella fascia oraria (in base alla tabella IO\_INTERVALLO\_INVIO, come già spiegato). Non c’è alcun filtro sul tipo di invio, e vengono inviati tutti senza distinzione. L’unico controllo è sulla presenza del particolare id\_coda nella tabella di esiti. Per i servizi con il campo “ripetibile” a MAI, non viene eseguito alcun reinvio se la riga ha già un esito corrispondente. Per i servizi con il campo “ripetibile” a ERRORE, il reinvio viene effettuato solo se l’ultimo esito in tabella ha un codice di errore 429 o 500. Per i servizi con il campo “ripetibile” a SEMPRE, il reinvio viene infine effettuato senza riguardo al tipo di esito in tabella.

I dettagli dei servizi sono contenuti nello swagger AppIO\_Sender\_Scheduler.yaml

Gli endpoint di uso generale sono i seguenti:

GET: [/appio/sender/timer](http://192.168.154.109:8080/LD_ComGE_Batch_AppIO/comge/batch/appio/timer)

Per avere la lista di tutti i timer attivi.

DELETE: [/appio/sender/timer?id=”timer\_name](http://192.168.154.109:8080/LD_ComGE_Batch_AppIO/comge/batch/appio/timer?id=)”

Per cancellare un timer chiamato “timer\_name”.

POST: [/appio/sender/timer](http://192.168.154.109:8080/LD_ComGE_Batch_AppIO/comge/batch/appio/timer)

Per creare un nuovo timer.

Il body della chiamata ha la forma:

{ "cron\_schedule": "schedulazione in formato cron",

"initial\_offset\_ms": 5000,

"timer\_name": "timer\_name"}

Il campo timer\_name è univoco ed identifica il timer.

Per il formato cron vedere il documento ufficiale: <http://www.quartz-scheduler.org/documentation/quartz-2.3.0/tutorials/crontrigger.html>

**FILE DI PROPERTIES**

Come tutti i progetti Spring Boot, il sender ha bisogno di un file di configurazione da creare sotto la cartella /conf del tomcat con il nome: SenderScheduler.properties. Tale file deve contenere i dettagli del DB dove sono presenti le tabelle di AppIO e i parametri per invocare l’integrator di WSO2 (modulo **ClientDispatcher)**. Nella cartella Properties del progetto viene riportato un esempio.