

Implementazione PMPAY

Piano tecnico di integrazione nel gateway PagoPA esistente

L'integrazione riutilizza l'architettura esistente: `PagopaController` → `PagopaService` → `IFornitorePagopaService` → adapter del fornitore. Servono principalmente un `PmPayService`, un client REST dedicato, i mapper, la gestione del token e i relativi test.

Copertura prevista

Funzionalità applicativa	Modalità PmPay	API PmPay
Inserimento dovuto	Nativa	POST <code>/ente/{codiceEnte}/pagamento</code>
Modifica dovuto	Nativa	PUT <code>/ente/{codiceEnte}/pagamento/{id}</code>
Annullamento dovuto	Nativa	DELETE <code>/ente/{codiceEnte}/pagamento/{id}</code>
Download avviso	Nativa	GET <code>/ente/{codiceEnte}/pagamento/{iuv}/bollettino</code>
Redirect pagamento	Nativa	POST <code>/ente/{codiceEnte}/pagamento/{iuv}/urlpagamento</code>
Verifica esito	Nativa	GET <code>/ente/{codiceEnte}/pagamento/{id}</code>
Marca da bollo digitale	Emulata/composita	Inserimento con <code>datiBollo</code> , poi URL pagamento
Import massivo	Emulata inizialmente	Ciclo di operazioni puntuali
Export pagati	Emulata	Ricerca pagamenti o interrogazione elenco IUUV

La specifica espone API batch native, ma per la prima versione è preferibile non usarle: il contratto comune consente azioni miste nello stesso flusso, mentre PmPay accetta una sola azione per batch; inoltre il formato del file di esito non è documentato in modo sufficiente.

1. Chiarire i mapping non definiti

1. Confermare se `Creato.code` contiene lo IUUV.
2. Verificare se `codiceEnte` coincide con il codice fiscale o è assegnato da PmPay.
3. Confermare se `tipoDovuto` può essere usato direttamente come `Versamento.servizio`.
4. Definire la destinazione di `datiSpecificiRiscossione: vociVersamento`, dati aggiuntivi o nessun invio.
5. Identificare la data effettiva del pagamento.

6. Confermare il mapping `CON` → `ESEGUITO` e degli altri stati verso `NON_ESEGUITO`.
7. Confermare se modifica ed eliminazione possono usare `idDebito`.
8. Definire il codice servizio per la marca da bollo.
9. Ottenere il formato preciso del file di esito batch.

2. Estendere la configurazione

Aggiungere all'enum dei fornitori:

```
PMPAY("PmPay", "PagoPA REST v1")
```

Aggiungere alla configurazione ente:

```
private String codiceEnteFornitore;  
private String codiceServizioBollo;  
private String urlRedirectOk;  
private String urlRedirectKo;  
private String urlRedirectCancel;  
private String urlNotificaS2S;
```

Riutilizzare `username`, `password`, `urlServiziPagamento` e `codiceServizio`. Le credenziali devono provenire da variabili d'ambiente o secret esterni.

```
yamme:  
  enti:  
    pmpay_collaudo:  
      id: comune_collaudo  
      fornitore: PMPAY  
      codiceEnteFornitore: "CODICE_ENTE_PMPAY"  
      codiceServizio: "MENSA"  
      codiceServizioBollo: "BOLLO"  
      username: "${PMPAY_USERNAME}"  
      password: "${PMPAY_PASSWORD}"  
      urlServiziPagamento: "https://service.pmpay.it/pagoparest"  
      urlRedirectOk: "https://app.example.it/pagamento/ok"  
      urlRedirectKo: "https://app.example.it/pagamento/ko"  
      urlRedirectCancel: "https://app.example.it/pagamento/annullato"  
      urlNotificaS2S: "https://app.example.it/pagamento/notifica"
```

3. Separare avvio flusso e polling

Il contratto corrente usa la stessa funzionalità per l'avvio di import/export e per la verifica dello stato. Questo è già incoerente per Siscom. Aggiungere:

```
VERIFICA_STATO_IMPORTAZIONE_FLUSSO_DOVUTI,  
VERIFICA_STATO_ESPORTAZIONE_FLUSSO_PAGATI
```

Aggiornare `PagopaService`, `MypayService`, `SiscomService` e il nuovo `PmPayService`. `MyPay` dichiarerà il polling nativo; `Siscom` e `PmPay` sincrono lo dichiareranno non presente.

4. Creare i modelli interni PmPay

Creare il package `it.yamme.pagopa.clientRest.pmpay.dto` con i soli modelli necessari:

- `PmPayCredenziali` e `PmPayToken`;
- `PmPayPagamento`, `PmPayVersamento`, `PmPayDatiDebitore` e `PmPayDatiBollo`;
- `PmPayPagamentoCreato`;
- `PmPayUrlPagamentoRequest/Response`;
- `PmPayRicercaPagamentoRequest/Response` e paginazione;
- `PmPayErrorResponse`.

È preferibile un client manuale limitato alle API usate, evitando di generare tutte le API amministrative presenti nella specifica.

5. Implementare autenticazione e client REST

Creare:

```
clientRest/PmPayClient.java
clientRest/PmPayTokenProvider.java
clientRest/PmPayApiException.java
```

Token provider

1. Chiamare `POST /autenticazione`.
2. Leggere token e scadenza.
3. Memorizzare il token per ente.
4. Rinnovarlo poco prima della scadenza.
5. Sincronizzare il rinnovo per evitare autenticazioni concorrenti.
6. Invalidarlo e riprovare una sola volta dopo un `401`.
7. Non scrivere credenziali o token nei log.

Client

Esporre metodi per creare, leggere, modificare ed eliminare pagamenti, scaricare il bollettino, creare l'URL di pagamento e ricercare pagamenti. Usare il `RestTemplate` già presente, aggiungendo timeout di connessione e risposta centralizzati.

6. Creare PmPayService

Creare `service/PmPayService.java`, implementando `IFornitorePagopaService` e annotandolo con `@Service`. Spring lo registrerà automaticamente nella lista degli adapter usata da `PagopaService`.

IMPORTA_DOVUTO	→ NATIVA
MODIFICA_DOVUTO	→ NATIVA
ANNULLA_DOVUTO	→ NATIVA
SCARICA_AVVISO_PAGAMENTO	→ NATIVA
REDIRECT_PAGAMENTO	→ NATIVA
VERIFICA_ESITO_PAGAMENTO	→ NATIVA
PAGA_MARCA_BOLLO_DIGITALE	→ EMULATA
IMPORTAZIONE_FLUSSO_DOVUTI	→ EMULATA
ESPORTAZIONE_FLUSSO_PAGATI	→ EMULATA

7. Mappare il dovuto comune

Creare `mapper/pmpay/PmPayPagamentoMapper.java`.

DTO comune	PmPay
iud	idDebito
codIuv	iuv , omesso quando va generato
dataScadenza	dataScadenza
codice identificativo pagatore	idFiscaleDebitore
tipo identificativo	datiDebitore.idType
nominativo/ragione sociale	datiDebitore.fullName
email	datiDebitore.email
importoDovuto	versamenti[0].importo
causaleVersamento	versamenti[0].causale
tipoDovuto	versamenti[0].servizio

8. Operazioni puntuali

Inserimento

1. Validare i dati comuni.
2. Costruire `PmPayPagamento`.
3. Eseguire il POST.
4. Leggere lo IUV dalla risposta o recuperarlo tramite IUD.
5. Restituire `ImportaDovutoResponseDTO`.

Modifica

1. Scegliere IUV o IUD.
2. Usare `?idDebito=true` quando necessario.

3. Costruire `ModificaPagamento`.
4. Eseguire il PUT e normalizzare l'esito.

L'API di modifica non consente di aggiornare tutti i dati dell'inserimento: debitore e servizio non sono presenti nello schema di modifica.

Annullamento

Eseguire il DELETE per IUV o IUD e considerare riuscita la risposta `204 No Content`.

9. Download dell'avviso

Chiamare GET `/ente/{codiceEnte}/pagamento/{iuv}/bollettino` e restituire i byte PDF nell'attuale `ScaricaAvvisoPagamentoResponseDTO`. Per PmPay non servono URL temporaneo e request token.

10. Redirect di pagamento

PmPay richiede `urlOk`, `urlKo`, `urlCancel` e, opzionalmente, `urlS2S` e `idRichiesta`. Aggiungere un oggetto opzionale `DatiRedirectPagamentoDTO` a `VerificaAvvisoRequestDTO` e alla richiesta di bollo. Se assente, usare i valori configurati sull'ente. MyPay può ignorare il nuovo oggetto.

Chiamare POST `/ente/{codiceEnte}/pagamento/{iuv}/urlpagamento` e mappare `location` in `urlRedirect`. PmPay non restituisce un vero `idSession`, quindi va lasciato nullo salvo diversa indicazione del fornitore.

11. Verifica dell'esito

Ricerca per IUV oppure per IUD con `?idDebito=true`. L'`idSession` non è utilizzabile da solo. Creare `mapper/pmpay/EsitoPagamentoMapper.java` e mappare inizialmente `CON` in `ESEGUITO`, gli altri stati in `NON_ESEGUITO`, dopo conferma del fornitore.

Per i pagamenti eseguiti valorizzare IUV, importo totale, IUD, importi dei versamenti, data pagamento ed eventuale ricevuta telematica.

12. Marca da bollo digitale

1. Creare un pagamento con un solo versamento.
2. Valorizzare `datiBollo.tipoBollo = "01"`, impronta e provincia.
3. Inserire il pagamento usando il servizio configurato per il bollo.
4. Recuperare lo IUV.
5. Richiedere l'URL di pagamento.
6. Restituire IUV e URL nel DTO comune.

13. Import massivo

Per la prima versione riutilizzare la strategia Siscom:

1. Scorrere tutte le righe.
2. Eseguire inserimento, modifica o annullamento secondo `azione`.
3. Continuare anche dopo gli errori puntuali.
4. Raccogliere la mappa IUD–IUV delle righe riuscite.
5. Restituire lo stato sincrono incorporato.
6. Generare un token sintetico `PMPAY-<UUID>` per uniformità.

In una fase successiva il batch nativo richiederà suddivisione per azione, token composto, aggregazione stati, parser dell'esito e regole di persistenza.

14. Export dei pagati

Supportare entrambi i filtri esistenti:

- elenco IUV: interrogazione puntuale;
- intervallo date: ricerca paginata con stato `CON`.

L'export sarà sincrono ed emulato: leggere tutte le pagine, mantenere i pagamenti conclusi, convertirli nel DTO comune e restituire `ESEGUITO` oppure `NESSUN_DOVUTO_TROVATO`.

15. Gestione degli errori remoti

Introdurre `FornitorePagopaException` con fornitore, operazione, stato HTTP, codice, descrizione sicura e indicazione di errore temporaneo. Aggiornare il gestore globale:

- `400`: richiesta applicativa non valida;
- `409`: operazione incompatibile con lo stato;
- `502`: errore o risposta non valida del fornitore;
- `504`: timeout.

16. Test

Aggiungere:

```
clientRest/PmPayClientTest.java
clientRest/PmPayTokenProviderTest.java
service/PmPayServiceTest.java
mapper/pmpay/PmPayPagamentoMapperTest.java
mapper/pmpay/EsitoPagamentoMapperTest.java
```

Verificare autenticazione, cache e rinnovo token, retry su 401, header, URL e query, mapping, generazione IUV, operazioni puntuali, PDF, redirect, bollo, stati, import con errori parziali, export paginato, timeout e assenza di segreti nei log. Per i test HTTP usare `MockRestServiceServer`.

17. Documentazione e collaudo

Aggiungere `docs/specifiche_pmpay.adoc`, esempio di configurazione, richieste Postman, aggiornamenti a README e revisioni, tabella delle funzionalità e procedura di collaudo.

Eseguire:

```
./mvnw test
./mvnw clean verify
```

Il collaudo in preproduzione deve coprire autenticazione, inserimento, download bollettino, URL pagamento, verifica stato, modifica, annullamento, import/export emulati e marca da bollo digitale.

Conclusione

Non servono nuovi endpoint pubblici per le operazioni principali. Il lavoro sostanziale riguarda client REST, adapter `PmPayService`, mapper, token e test. Le modifiche consigliate al contratto comune sono l'aggiunta dei dati di redirect e la separazione tra avvio flusso e polling dello stato.