

# Documentation technique — Backend Apex iDEM Connect



**iDEM Connect** : FAI international (CA 1,8 Md€, 3 950 salariés). Objectif : outiller la force de vente via Salesforce pour gérer ventes, clients et contrats.

## Objets Salesforce :

- **Standards** : Account, Contract, Order, OrderItem, Task
- **Custom** : Account.Active\_\_c (Checkbox), Task.Auto\_Created\_\_c (Checkbox)

## Règles métier :

- **RG-01** : Bloquer l'activation d'un Order sans OrderItem .
- **RG-02** : Maintenir Account.Active\_\_c à true à l'insertion d'un Order , à false à la suppression du dernier.
- **RG-03** : Chaque 1er lundi à 09:00, créer une Task « Call » (J+5) pour chaque Account sans Order ni « Call » récente.

## Configuration Org :

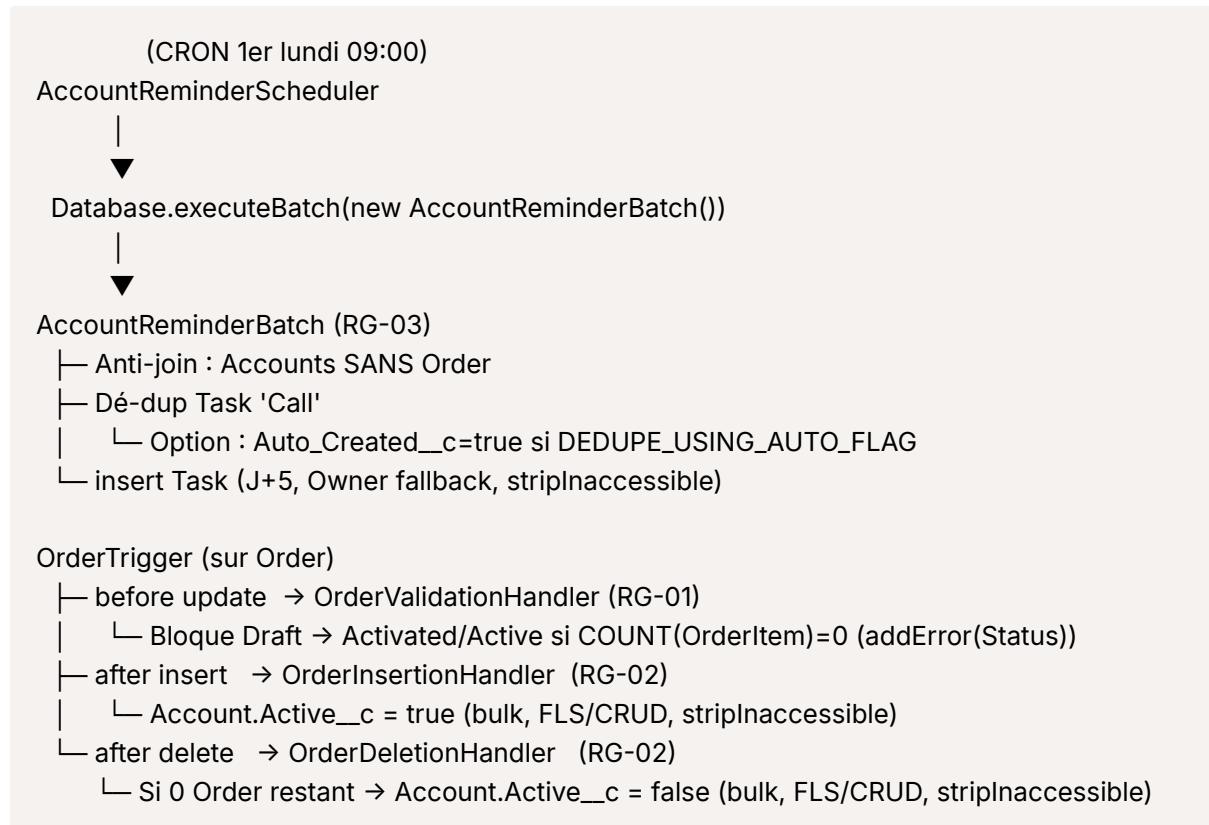
- **Picklists** : Order.Status = {Draft, Activated, Cancelled} , Task.Subject = Call , Task.Status = Nouvelle , Task.Priority = Normal
- **Activity Reminders** : activés
- **Validation Rule** : Order requiert un ContractId non vide
- **Mentor** : pas de Permission Set, pas de Custom Metadata — configuration via AppConfig.cls .

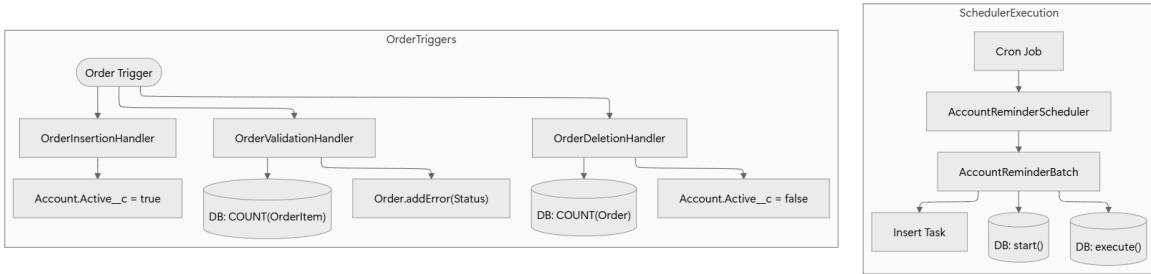
## Table des matières

1. Vue d'architecture & traçabilité
2. **OrderTrigger.trigger**
3. **OrderValidationHandler.cls**
4. **OrderInsertionHandler.cls**
5. **OrderDeletionHandler.cls**
6. **AccountReminderBatch.cls**
7. **AccountReminderScheduler.cls**
8. **AppConfig.cls**
9. **TestDataFactory.cls**
10. **Classes de test unitaires (synthèse)**
11. Sécurité, performances & bonnes pratiques (récap)
12. Recommandations & évolutions
13. Résumé couverture & exécution des tests

## 1) Vue d'architecture & traçabilité

### Diagramme (haut niveau)





## Matrice de traçabilité (exigences → artefacts → tests)

ID	Exigence (résumé)	Implémentation	Tests principaux
RG-01	Interdire activation d'un Order sans OrderItem	OrderTrigger.before update → OrderValidationHandler.handleStatusValidation()	OrderValidationHandlerTest
RG-02	Insertion Order → Account.Active__c = true ; suppression du dernier Order → Active__c = false	OrderTrigger.after insert → OrderInsertionHandler.handlePostInsert() ; OrderTrigger.after delete → OrderDeletionHandler.handlePostDelete()	OrderInsertionHandlerTest , OrderDeletionHandlerTest
RG-03	1er lundi 09:00 : créer Task 'Call' (J+5) pour Accounts sans Order et sans Call existante	AccountReminderScheduler (CRON) → AccountReminderBatch	AccountReminderBatchTest , AccountReminderSchedulerTest

| i VR ContractId sur Order (existant) : garantit l'ancre Contract→Order côté UI/données.

## 2) Fiche technique — OrderTrigger.trigger

### 2.1 Fiche signalétique

Élément	Détail
Nom	OrderTrigger
Objet	Order
Événements	before update , after insert , after delete
Règles	RG-01, RG-02
Rôle	Orchestration (trigger léger)
DML/SOQL	✗ Aucun (délégué aux handlers)

### 2.2 Comportement

- **before update** → `OrderValidationHandler.handleStatusValidation(Trigger.new, Trigger.oldMap)` (RG-01)
- **after insert** → `OrderInsertionHandler.handlePostInsert(Trigger.new)` (`Active__c = true`)
- **after delete** → `OrderDeletionHandler.handlePostDelete(Trigger.old)` (`Active__c = false si plus d'Orders`)

## 2.3 Bonnes pratiques

- **1 trigger / objet, zéro** logique métier dans le trigger.
- Bulk-safe (List/Map), testabilité via handlers.

## 3) Fiche technique — **OrderValidationHandler.cls**

### 3.1 Fiche signalétique

Élément	Détail
Type	Handler (logic RG-01)
Signature	<code>public inherited sharing class OrderValidationHandler</code>
Méthode	<code>handleStatusValidation(List&lt;Order&gt; newOrders, Map&lt;Id, Order&gt; oldOrderMap)</code>
DML	✗ Aucun (lecture + <code>addError()</code> )
Sécurité	Lecture seule

### 3.2 Description fonctionnelle

Empêche l'**activation** d'un Order **sans produits** : lorsque le **Status passe de Draft à Activated**, on compte les `OrderItem`. S'il n'y en a **aucun**, `addError()` sur `Status` (bloquant).

### 3.3 Logique (pas à pas)

1. Détecter transition **Draft → Activated**.
2. SOQL agrégé `COUNT(OrderItem)` groupé par `OrderId`.
3. `addError` pour les Orders avec `0` item.

### 3.4 Limite connue (assumée)

Non-bloquant volontaire : la détection cible Draft→Activated.

Les transitions **(Cancelled/Autre) → Activated** ne sont **pas** interceptées pour le moment (patch possible ultérieurement).

### 3.5 Tests

- KO : activation sans items.
- OK : activation avec  $\geq 1$  item.
- Mix bulk.

## 4) Fiche technique — **OrderInsertionHandler.cls**

### 4.1 Fiche signalétique

Élément	Détail
Type	Handler (RG-02+)
Signature	public inherited sharing class OrderInsertionHandler
Méthode	handlePostInsert(List<Order> newOrders)
DML	<input checked="" type="checkbox"/> update Account (bulk)
Sécurité	Security.stripInaccessible(UPDATABLE)

## 4.2 Description fonctionnelle

Au moment de l'**insertion d'un Order**, on **coche** `Account.Active__c = true` pour tous les **Accounts concernés**.

## 4.3 Logique

- Collecte des `AccountId` des Orders insérés.
- Prépare une liste d' `Account(Id, Active__c=true)` .
- FLS/CRUD** avec `stripInaccessible` puis `update` en masse.

## 4.4 Tests

- Vérifie `Active__c` **true** après insertion de l'Order.
- Guards **null/empty** couverts.

# 5) Fiche technique — OrderDeletionHandler.cls

## 5.1 Fiche signalétique

Élément	Détail
Type	Handler (RG-02)
Signature	public inherited sharing class OrderDeletionHandler
Méthode	handlePostDelete(List<Order> deletedOrders)
DML	<input checked="" type="checkbox"/> update Account (bulk)
Sécurité	isUpdateable(Active__c) + stripInaccessible(UPDATABLE)

## 5.2 Description fonctionnelle

Après **suppression** d'Orders, si un **Account n'a plus aucun Order**, alors `Active__c = false` .

## 5.3 Logique

- Extraire `AccountId` depuis `Trigger.old` .
- `COUNT(Orders)` restants par compte.
- Pour ceux avec `0` restant → `Active__c=false` (bulk update + FLS).

## 5.4 Tests

- Cas **tous supprimés** → `Active__c=false` .

- Cas **partiel** → `Active__c` reste **true**.
- Early return couvert.

## 6) Fiche technique — **AccountReminderBatch.cls**

### 6.1 Fiche signalétique

Élément	Détail
Type	<code>global class</code> ; <code>Database.Batchable&lt;SObject&gt;</code>
Sharing	<code>with sharing</code>
Objet cible	<code>Account</code>
Règle	<code>RG-03</code>
DML	<code>insert Task</code>
Sécurité	<code>Task.isCreateable()</code> ; <code>Security.stripInaccessible(CREATABLE)</code> ; vérifs <b>CRUD/FLS</b> lecture ( <code>Task.WhatId</code> , <code>Task.Subject</code> , <code>Task.Auto_Created__c</code> )
Point d'entrée	<code>AccountReminderScheduler</code> ( <code>Schedulable</code> ) — CRON <code>0 0 9 ? * MON#1</code>
Taille de lot (exécution planifiée)	200

### 6.2 Description fonctionnelle

Crée une `Task` "Call" (Status "Nouvelle", Priority "Normal/Normale", rappel J+5) pour chaque `Account` sans `Order` et sans aucune `Task` "Call" existante (qu'elle soit manuelle ou auto).

- **Owner** : `Account.OwnerId` ; *fallback* = utilisateur courant si owner **absent/inactif**.
- **Marquage** : `Task.Auto_Created__c = true`.
- **Dates** : `ActivityDate = today() + 5` ; `IsReminderSet = true` ; `ReminderDateTime = now() + 5j`.
- **Option de déduplication** : si `AppConfig.DEDUPE_USING_AUTO_FLAG = true`, la dédup ne considère **que** les tâches marquées `Auto_Created__c = true`.

### 6.3 Logique

- `start(...)` — `Database.QueryLocator`
  - **Anti-join** : sélection des `Account` sans `Order` :
    - `SELECT Id, OwnerId FROM Account WHERE Id NOT IN (SELECT AccountId FROM Order WHERE AccountId != null) WITH SECURITY_ENFORCED`
  - *Projection minimale* ( `Id` , `OwnerId` ) ; `WITH SECURITY_ENFORCED` utilisé ici (requête simple).
- `execute(...)` — **(dé-dup + insert)**
  1. **Préparation scope** : `Set<Id>` des `Account` à traiter.
  2. **Dé-dup "Call"** (agrégat `Task` → `GROUP BY WhatId`) :
    - **Mode par défaut** : existe **toute** `Task` avec `Subject = 'Call'` → **skip**.
    - **Mode auto-only (si flag activé)** : existe `Task` `Subject='Call'` et `Auto_Created__c = true` → **skip**.

- ⚠ Pas de `WITH SECURITY_ENFORCED` sur l'agrégat → vérifs CRUD/FLS explicites **avant requête** :

`Task.isAccessible()` + accès champs `WhatId` / `Subject` / `Auto_Created__c` si mode auto.

3. **Contrôle de création** : `Task.isCreateable()` (*guard global*).

4. **Construction des `Task`** (bulk) :

- `OwnerId` = owner actif sinon **fallback** `UserInfo.getUserId()`.
- Champs : `WhatId = Account.Id`, `Subject='Call'`, `Status='Nouvelle'`, `Priority='Normal'`, `IsReminderSet=true`,  
`ReminderDateTime=now() + 5j`, `ActivityDate=today() + 5`, `Auto_Created__c=true`.
- Type : facultatif via `AppConfig.TASK_TYPE_OPT` (si non vide).

5. **Sécurisation FLS** : `Security.stripInaccessible(AccessType.CREATABLE, records)` → retrait des champs non créables.

6. **Insertion résiliente** : `Database.insert(..., false)` + **logging détaillé** des `SaveResult` en cas d'échecs partiels.

- `finish(...)`

◦ Hook de fin léger (journalisation simple). *Extensible* (ex : email/PE).

## 6.4 Paramètres (constantes `AppConfig`)

- `TASK SUBJECT = 'Call'`
- `TASK STATUS = 'Nouvelle'`
- `TASK_PRIORITY = 'Normal'` (*libellé FR : "Normale"*)
- `REMINDER_DAYS = 5`
- `TASK_TYPE_OPT = 'Call'` (*peut être vide ; alors non renseigné*)
- `@TestVisible Boolean DEDUPE USING AUTO FLAG = false` (**par défaut**)

## 6.5 Tests

- Crée une `Task` auto si **Account sans Order et sans Call**.
- Ignore si `Call` existante (manuelle ou auto).
- Ignore si `Account` a un `Order`.
- J+5 sur `ActivityDate` et `ReminderDateTime`.
- Owner fallback si owner inactif/absent.
- Bulk : 250 comptes → 250 tâches (batch size ≥ 250 en test).
- Flag **auto-only** : présence d'une **auto-task** existante  **bloque** la création.
- **stripInaccessible(CREATABLE)** : insertion **même si** certains champs ne sont pas créables.
- Exécution type **scheduler** (appel direct du batch sous `Test.startTest()/stopTest()`).

Couverture mesurée (classe) : 86 % — org-wide 90 % (23 tests, 100 % pass).

## 7) Fiche technique — AccountReminderScheduler.cls

### 7.1 Fiche signalétique

Élément	Détail
Type	global class , Schedulable
Rôle	Déclencher AccountReminderBatch selon CRON
Helper	CRON_9AM_FIRST_MON() ⇒ '0 0 9 ? * MON#1'

### 7.2 Description fonctionnelle

Planifie l'exécution **mensuelle** (1er lundi, 09:00) du batch de rappel d'appels (RG-03).

### 7.3 Comportement

- `execute()` → `Database.executeBatch(new AccountReminderBatch(), 200)`
- **Planification interface** : *Setup* → *Apex Classes* → *Schedule Apex*
  - Job Name : `AccountReminderScheduler`
  - Frequency : **Weekly**, Day = **Monday**, Time = **09:00**
  - (CRON interne de référence : `0 0 9 ? * MON#1`)

### 7.4 Tests

- Vérifie qu'un **JobId** est retourné à la planification ;
- Couvre **helper CRON** et `execute()` direct.

## 8) Fiche technique — AppConfig.cls

### 8.1 Fiche signalétique

Élément	Détail
Type	Classe de configuration (constantes)
Rôle	Centraliser valeurs système & métier

### 8.2 Contenu (extrait)

- `ACTIVATION_STATUSES_LC = {'activated','active'}` (*pour RG-01 FR/EN*)
- `TASK_SUBJECT='Call'`,
- `TASK_STATUS='Nouvelle'`,
- `TASK_PRIORITY='Normal'`,
- `REMINDER_DAYS=5`
- `DEDUP_USING_AUTO_FLAG : Boolean` (*option*)
- `TASK_TYPE_OPT : String` (*optionnel*) + `@TestVisible TASK_TYPE_OPT_OVERRIDE`

## 9) Fiche technique — **TestDataFactory.cls** ( `@isTest` )

### 9.1 Fiche signalétique

Élément	Détail
Type	Utilitaire de test
Responsabilité	Générer/insérer/nettoyer des données (Accounts, Contracts, Products, PBE, Orders, OrderItems, Tasks, Users)
Patron	<code>generateX()</code> (sans DML) / <code>createX()</code> (avec DML)

### 9.2 Points clés

- Utilise `Test.getStandardPricebookId()` pour PBE.
- Méthodes `runAs` & `createTestUser` prêtes si besoin.
- `deleteTestData()` pour nettoyage ciblé (appelée déjà en test).

## 10) Synthèse des tests unitaires

- `AccountReminderBatchTest`
  - Scénarios :
    - Création Task si **No Order & No Call**.
    - Skip si **Call manuelle OU auto** existe (ouverte/fermée/ancienne).
    - Skip si **Account a un Order**.
    - **Fallback Owner** actif.
    - **Dates J+5** (Activity/Reminder).
    - **Bulk 250**.
    - **Scheduler-like** (batch direct).
    - **Dédouble stricte** (`DEDUPE_USING_AUTO_FLAG=true`).
    - **stripInaccessible** (chemin d'insertion partielle).
- `AccountReminderSchedulerTest` : helper CRON + `execute()`.
- `OrderDeletionHandlerTest` : désactivation compte si 0 Order restant ; cas partiel.
- `OrderInsertionHandlerTest` : activation compte à l'insert ; guards ; logs d'erreurs.
- `OrderValidationHandlerTest` : KO sans items ; OK avec items ; cas mixte (save results).
- `TestDataFactory` : nettoyage global.

## 11) Sécurité, performances & bonnes pratiques (récap)

- **Architecture** : 1 trigger / objet ; logique en **handlers** ; Batch/Scheduler séparés.
- **Bulkification** : Set/Map, agrégats, DML hors boucles.
- **SOQL** : projection minimale ; agrégats `GROUP BY` ; anti-join stable au `start()`.

- **Gouvernance** : `QueryLocator` (volumes), `allOrNone=false` (résilience), `Test.startTest/stopTest`.
- **Sécurité** : `WITH SECURITY_ENFORCED` au `start()` ; `isCreateable / isUpdateable` ; `Security.stripInaccessible`.
- **UX** : rappel J+5 (`isReminderSet=true` + `ReminderDateTime`).
- **Testabilité** : Factory + overrides contrôlés.

## 12) Résumé exécution des tests & couverture (dernier run)

Outcome: Passed | Tests Ran: 23 | Pass Rate: 100% | Org-Wide Coverage: 90%

Coverage by class:

- OrderTrigger 100%
- AppConfig 100%
- AccountReminderScheduler 100%
- OrderValidationHandler 97%
- OrderDeletionHandler 95%
- OrderInsertionHandler 91%
- AccountReminderBatch 86%

## 13) Déploiement & exploitation

### A. SFDX — déploiement (CLI)

```
# Auth vers l'org
sfdx auth:web:login -a DevOrg
sfdx force:config:set defaultusername=DevOrg

# Déployer tout le code
sfdx force:source:deploy -p force-app/main/default
```

### B. Planification du scheduler

UI : Setup → Apex Classes → Schedule Apex

- **Job Name** : `AccountReminderScheduler`
- **Fréquence** : Weekly → **Monday**
- **Heure** : **09:00**

| CRON interne : 0 0 9 ? \* MON#1 (1er lundi du mois)

**Apex (optionnel) :**

```
System.schedule('AccountReminderScheduler', '0 0 9 ? * MON#1', new AccountReminderScheduler());
```

## C. Vérifications d'exploitation (evidence pack)

- Scheduled Jobs → présence du job `AccountReminderScheduler` (prochaine exécution = 1er lundi 09:00)
- Exécution **manuelle** de `AccountReminderBatch` (depuis "Execute Anonymous") pour smoke test : création de 1-2 Tasks "Call" attendue sur des comptes sans orders ni calls.
- Logs d'erreurs (éventuels) dans *Debug Logs*.

## D. Commande : Tests & couverture

### Exécution (CLI)

```
# 1) Lancer tous les tests locaux
sfdx force:apex:test:run -l RunLocalTests -r human -c -w 20

# 2) Rapport lisible
sfdx force:apex:test:report -i <TEST_RUN_ID> -r human > ./doc/TestReport.txt

# 3) Couverture JSON
sfdx force:apex:test:report -i <TEST_RUN_ID> -c > ./doc/coverage.json
sfdx force:apex:test:run -l RunLocalTests -r junit -c -w 20 -d ./doc/
```