010 — Itération (Sprint) : Développement CRUD Admin + Socle API

Projet : Refonte du SI DIGICHEESE

10.1. Contexte de l'itération

Cette itération correspond à la phase de démarrage du projet DIGICHEESE (TP7), réalisée en équipe de 4 personnes.
La stratégie retenue a été de **répartir le travail par lots fonctionnels** : chaque lot correspond à un **CRUD Admin** (M3 à M8), implémenté de bout en bout (models/repository/service/routes/tests/doc).

Les items **M0 à M2** constituent le socle (conventions, démarrage API, Swagger) et ont été traités en priorité afin d'harmoniser le développement de tous les CRUD.

10.2. Sprint Planning (planification)

Objectif du sprint

Livrer un backend exploitable et testable via Swagger, en priorisant :

- le socle API + conventions,
- les CRUD Admin (M3 à M8),
- les premiers tests automatisés pour sécuriser les routes.

Backlog ciblé

MUST — Socle

- M0 — Intégration & conventions communes (socle de standardisation)
- M1 — Socle API : FastAPI + MySQL + .env + lancement
- M2 — Swagger opérationnel + conventions endpoints

MUST — CRUD Admin

- M3 — CRUD Admin : Utilisateurs
- M4 — CRUD Admin : Communes
- M5 — CRUD Admin : Objets
- M6 — CRUD Admin : Conditionnements
- M7 — CRUD Admin : Poids
- M8 — CRUD Admin : Poids-vignettes

Stratégie d'exécution

- Découpage **par CRUD** : chaque développeur prend en charge **2 CRUD** complets.
- Standardisation : M0 sert de "contrat" commun (structure, conventions, erreurs, DoD).
- Validation progressive : implémentation → tests automatisés → correction → validation Swagger.

10.3. Organisation des rôles (projet)

Rôles projet (pour l'exercice TP)

- **PO / Scrum Master** : rôle assuré de manière collégiale (décisions partagées).
- Pour la formalisation attendue :
  - **PO/Scrum Master : Xavier (collégial)**
  - **Référent socle technique (M1/M2) : Stanislas**
  - **Équipe de dev : les 4 membres**

Répartition par lots CRUD

- Chaque membre a choisi **2 CRUD Admin** et les a implémentés en autonomie, en respectant les conventions partagées.
- Chaque CRUD comprend : `models` + `repository` + `service` + `routes` + `tests` + éléments Swagger.

Contributions spécifiques

- **Stanislas** : prise en charge principale de **M1** (socle API) et **M2** (Swagger + conventions).
- **Xavier** : en charge de la **rédaction** (première version du backlog/documentation fonctionnelle) et contribution au développement sur ses lots CRUD.
- **Imen**: coordination majoritaire de la **présentation finale** (support PowerPoint), avec contribution de toute l'équipe.
- **Thi Thu Hien**: support méthodologique sur l'usage des **branches Git** (bonne pratique pour éviter conflits).

10.4. Daily Meetings (simulation structurée)

> Format : Avancement / Blocages / Actions prévues

Daily — Jour 1 (cadrage et structuration)

- **Avancement :**
  - Prise en main du sujet et compréhension du périmètre.

- Démarrage de la structuration backlog (MoSCoW) et des conventions.

- **Blocages :**
  - Aucun avancement code (phase de cadrage).
  - Incertitudes sur certaines parties fonctionnelles issues des UML / CDC (risque de développer hors priorités).
- **Actions prévues :**
  - Définir les conventions (M0) et préparer la structure commune.
  - Établir le backlog et répartir les lots CRUD.

Daily — Jour 2 (démarrage développement)

- **Avancement :**
  - Début des implémentations CRUD (routes + services + repositories + modèles).
  - Mise en place progressive du socle API.
- **Blocages :**
  - Harmonisation des patterns entre développeurs (risque d'écarts entre CRUD).
- **Actions prévues :**
  - S'aligner sur les conventions M0.
  - Stabiliser le socle M1/M2 (Swagger / DB / .env).

Daily — Jour 3 (consolidation et stabilisation)

- **Avancement :**
  - Poursuite des CRUD.
  - Intégration continue des routes, validation via Swagger.
- **Blocages :**
  - Ajustements techniques sur la structure et le mapping erreurs.
- **Actions prévues :**
  - Finir les CRUD du périmètre MUST.
  - Préparer la phase tests automatisés.

Daily — Jour 4 (tests automatisés + debug)

- **Avancement :**
  - Développement des tests automatisés pour les routes.
  - Débugage / corrections suite aux résultats des tests et validations Swagger.
- **Blocages :**
  - Écarts ponctuels de mise en œuvre liés au niveau Python (rattrapés).
- **Actions prévues :**

- Stabiliser l'ensemble (routes + tests) et homogénéiser.

Daily — Jour 5 (finalisation livrables)

- **Avancement :**
  - Finalisation et relecture des livrables.
  - Consolidation de la présentation.
- **Blocages :**
  - Aucun bloquant majeur.
- **Actions prévues :**
  - Livrer la version finale + supports.
  - Préparer le prochain cycle (tests/campagnes + documentation Jira/Confluence).

10.5. Sprint Review (résultats)

La Sprint Review a permis de démontrer l'état réel du backend DIGICHEESE à la fin de l'itération, en comparant le backlog prévu et les fonctionnalités effectivement livrées.

10.5.1 Fonctionnalités livrées

Les éléments suivants ont été implémentés et sont démontrables :

- Socle de conventions (M0) validé et appliqué
- Socle API FastAPI + configuration DB (M1)
- Swagger opérationnel (M2)
- CRUD Admin développés par lots :
  - Utilisateurs
  - Objets
  - Conditionnements
  - Poids
  - Poids-vignettes
  - Communes (présent côté code)

Chaque CRUD est testable via Swagger et suit la même architecture technique.

Des tests automatisés ont été développés et exécutés pour sécuriser les routes principales.

10.5.2 Preuves de livraison

La démonstration s'appuie sur :

- Swagger accessible via `/docs`
- Endpoints exécutables en direct
- Tests automatisés (pytest)
- Structure du dépôt conforme à l'architecture annoncée

Le backend est exécutable et stable à la fin du sprint.

10.5.3 Résultats techniques

Implémentation type : CRUD Objet (référence architecture)

Le CRUD Objet a servi de modèle de référence pour l'ensemble des CRUD Admin.

Architecture respectée :

Router → Service → Repository → Model

- Router FastAPI : exposition HTTP + gestion erreurs 404
- Service : couche métier intermédiaire
- Repository : accès DB isolé + rollback sécurisé
- Schémas Pydantic : Create / Patch / Out
- Modèle SQLAlchemy conforme CDC

Bonnes pratiques appliquées :

- PATCH sécurisé avec liste blanche de champs modifiables
- rollback DB en cas d'erreur SQL
- séparation claire des responsabilités
- validation Pydantic
- gestion des rôles Admin via dépendance FastAPI
- Swagger généré automatiquement

Ce modèle a été répliqué pour les autres CRUD Admin.

10.5.4 État Jira vs état réel du code

Dans le cadre de la simulation de sprint :

- certains CRUD sont présents dans le code
- mais volontairement non marqués "Done" dans Jira

Cela permet de simuler une fin d'itération réaliste avec :

- backlog restant
- dette technique identifiée
- travail à reporter au sprint suivant

Exemple : le CRUD Communes est implémenté mais non clôturé dans Jira.

10.5.5 Éléments non livrés (hors scope ou reportés)

Les éléments suivants ne faisaient pas partie du périmètre de l'itération :

- IHM (front-end) → Swagger sert de simulation
- Sécurité avancée
- Domaine Op-stocks
- Extensions métier optionnelles

Un prototype IHM a été exploré hors sprint mais non intégré officiellement.

10.5.6 Points d'amélioration identifiés

La review met en évidence plusieurs axes d'amélioration :

- alignement des codes HTTP REST (DELETE → 204)
- harmonisation des tags Swagger
- ajout systématique des tests automatisés
- meilleure synchronisation intermédiaire entre développeurs
- versioning API (préfixe `/api/v1`)
- intégration plus précoce des tests dans le cycle de dev

Ces points constituent la base du plan d'action du sprint suivant.

10.6. Sprint Retrospective (processus et plan d'action)

Ce qui a bien fonctionné

- **Structuration en amont** (M0 + backlog) : a permis de corriger le tir avant de coder.
- **Priorisation MoSCoW** : a facilité les arbitrages et la répartition du travail.
- **Découpage par CRUD** : a permis un travail parallèle efficace.
- **Adoption des branches Git** : a limité les conflits et protégé le travail de chacun.
- **Collaboration** : communication rapide, respectueuse, avec feedbacks utiles.

Points de friction / limites identifiées

- Harmonisation : un point formel aurait dû être fait **après la modélisation** et après chaque grande étape.
- Tests : ajoutés plutôt a posteriori ; l'intégration "tests plus tôt" aurait amélioré l'efficacité.
- Montée en compétence Python : quelques écarts de mise en œuvre, corrigés au fil de l'eau.

Plan d'action (prochain sprint)

1. **Point d'alignement obligatoire** après :
   - modélisation DB,
   - définition des schémas,
   - conventions erreurs,
   - inclusion routers.
2. **Tests dès le début** (smoke + tests CRUD minimal) pour limiter le debug tardif.
3. **Formaliser une check-list d'intégration** (DoD + conventions) avant merge.
4. Lancer la phase suivante :
   - campagnes de tests (outillage)
   - documentation Jira/Confluence (travail actuel)
   - amélioration de la couverture tests

10.7. Conclusion de l'itération

Cette itération a permis de construire une base exploitable :

- backlog structuré,
- socle API stabilisé,
- CRUD Admin réalisés en parallèle,
- tests automatisés et corrections intégrées,
- livrables finalisés (support de présentation).

Le projet est désormais prêt pour une phase de consolidation (tests/campagnes, documentation, qualité) et pour l'intégration des options si souhaité.