



Fromagerie Digicheese

Refonte du SI - TP7

Réalisé par :

Thi Thu Hien NGUYEN
Stanislas DELANNOY

Xavier DEGUERCY
Imen KHAMMASSI

Objectif : conception d'un socle applicatif structuré, fonctionnel et testable

21-01-2026

Version Beta V0



Plan

1. Contexte, Problématique métier & Objectif du projet
2. Analyse des besoins & Fonctionnalités implémentées
3. Organisation, pilotage & Priorisation MoSCoW
4. Architecture technique du backend & Modélisation logique des données
5. API : Structure & Endpoints & Gestion des règles métier et des erreurs
6. Tests automatisés et qualité logicielle
7. Outils et technologies utilisées, mise en place, exécution du projet & Résultats obtenus
8. Limites & Perspectives
9. Conclusion





Plan

- 1. Contexte, Problématique métier & Objectif du projet**
- 2. Analyse des besoins & Fonctionnalités implémentées**
- 3. Organisation, pilotage & Priorisation MoSCoW**
- 4. Architecture technique du backend & Modélisation logique des données**
- 5. API : Structure & Endpoints & Gestion des règles métier et des erreurs**
- 6. Tests automatisés et qualité logicielle**
- 7. Outils et technologies utilisées, mise en place, exécution du projet & Résultats obtenus**
- 8. Limites & Perspectives**
- 9. Conclusion**





Plan

1. Contexte, Problématique métier & Objectif du projet

1.1. Contexte du projet

1.2. Périmètre présenté

1.3. Problématique métier

1.4. Objectif du projet



1. Contexte, Problématique métier & Objectif du projet

Contexte

Périmètre présenté

Problématique métier

Objectif du projet

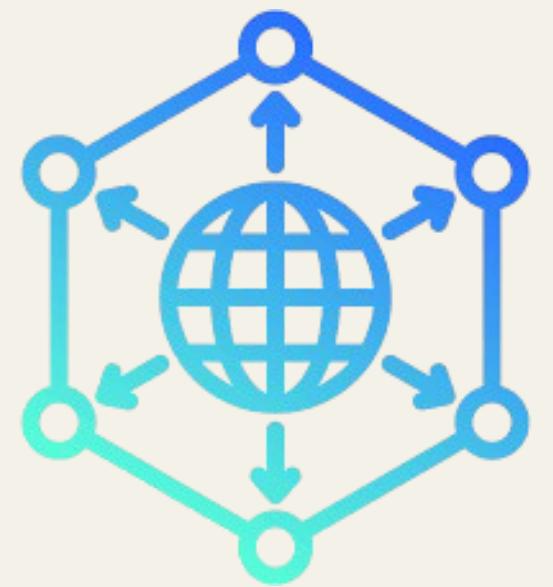


Contexte du projet

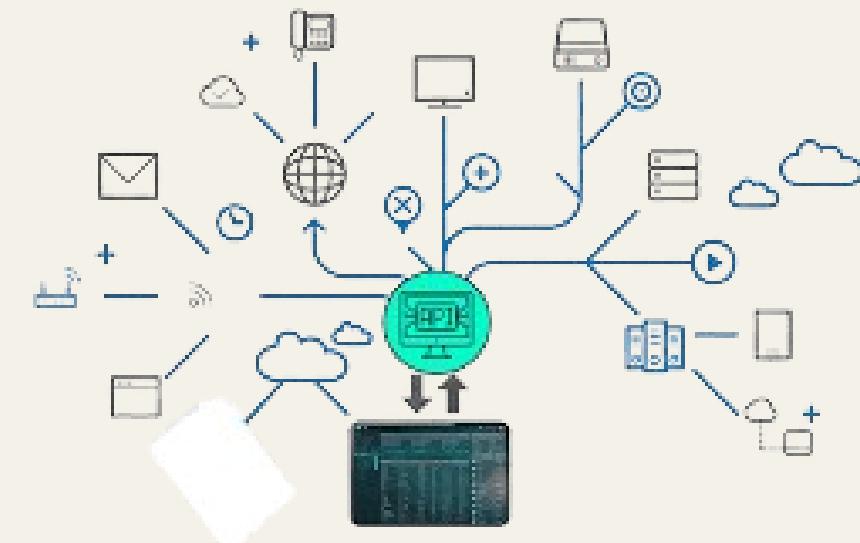
L'entreprise DIGICHEESE est une fromagerie disposant :



D'une activité de vente aux particuliers



D'un réseau de distribution vers des grandes surfaces



D'un besoin croissant de structuration de son système d'information



1. Contexte, Problématique métier & Objectif du projet

Contexte

Périmètre présenté

Problématique métier

Objectif du projet



Périmètre

Périmètre présenté

- CRUD Admin sur tables métiers (ex. objets / conditionnements / communes / poids...)
- (Option) endpoints Clients (OP-colis)
- Swagger + scénarios de tests
- Suite de tests automatisés + traces d'exécution



1. Contexte, Problématique métier & Objectif du projet

Contexte

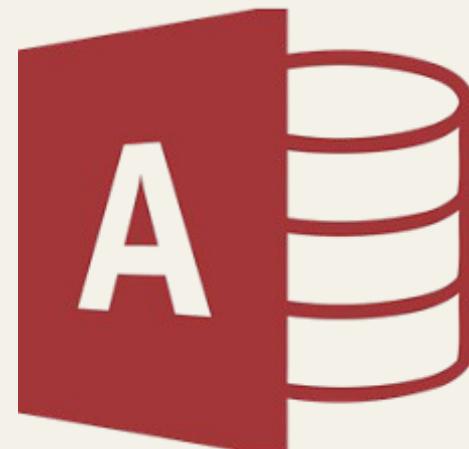
Périmètre présenté

Problématique métier

Objectif du projet



Problématique actuelle



Access 2000



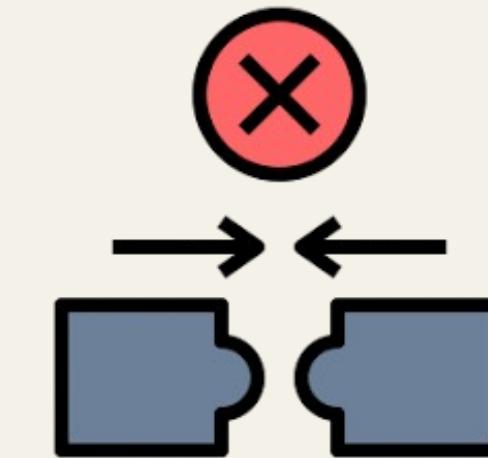
Obsolète



Instable



Non maintenable



Incompatible avec les systèmes récents



Navigation rigide



1. Contexte, Problématique métier & Objectif du projet

Contexte

Périmètre présenté

Problématique métier

Objectif du projet

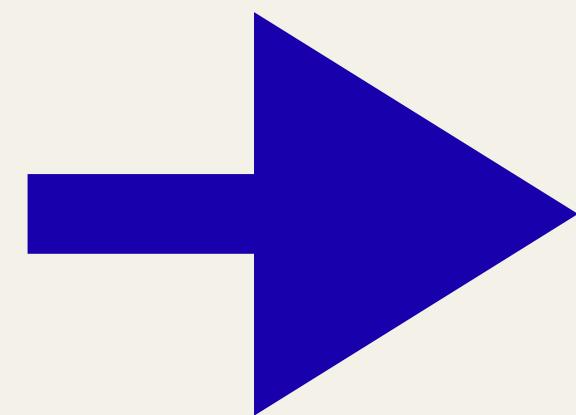


Objectifs :

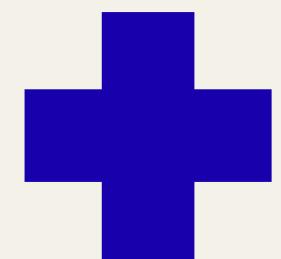
- Refonte complète : API + base MySQL (et front simulé via Swagger)
- Migrer vers une solution client léger/ Intranet



Solution
actuelle



Intranet



1. Contexte, Problématique métier & Objectif du projet

Contexte

Périmètre présenté

Problématique métier

Objectif du projet



L'objectif principal de la solution proposée est d'assurer et d'améliorer :



La fluidité



L'ergonomie



La maintenabilité



L'évolutivité





Plan

1. Contexte, Problématique métier & Objectif du projet
2. Analyse des besoins & Fonctionnalités implémentées
3. Organisation, pilotage & Priorisation MoSCoW
4. Architecture technique du backend & Modélisation logique des données
5. API : Structure & Endpoints & Gestion des règles métier et des erreurs
6. Tests automatisés et qualité logicielle
7. Outils et technologies utilisées, mise en place, exécution du projet & Résultats obtenus
8. Limites & Perspectives
9. Conclusion



Plan

2. Analyse des besoins & Fonctionnalités implémentées

2.1. Exigences fonctionnelles

2.2. Exigences techniques

2.3. Cadrage fonctionnel

2.4. Scénario nominal

2.3. Fonctionnalités implémentées



2. Analyse des besoins & Fonctionnalités implémentées

Exigences fonctionnelles

Exigences techniques

Cadrage

Scénario

Fonctionnalités

Exigences fonctionnelles :

Domaine	Exigences fonctionnelles
Général	Refonte du système existant/ Application interne (Intranet)
Sécurité/ Accès	Gestion par rôle Administrateur
Référentiels métier	CRUD des Roles, Utilisateurs, Clients, Objets, Conditionnements, Poids, Poids-Vignettes, Communes, Adresse.
Administrateur	Accès et gestion globale des données



2. Analyse des besoins & Fonctionnalités implémentées

Exigences fonctionnelles

Exigences techniques

Cadrage

Scénario

Fonctionnalités

Exigences techniques:

Domaine	Exigences techniques
Architecture	Backend API REST
Données	MYSQL (Base de données relationnelle)
Accès aux données	Opérations CRUD via API
Conception	Séparation des responsabilités (routes, services, accès données, repositories, models)



2. Analyse des besoins & Fonctionnalités implémentées

Exigences fonctionnelles

Exigences techniques

Cadrage

Scénario

Fonctionnalités

Qualité du code	Code structuré et maintenable
Documentation	Documentation de l'API (Swagger)
Tests	Tests automatisés unitaires de l'API (PyTest)
Déploiement	Projet exécutable localement
Versionning	Gestion du code avec Git



2. Analyse des besoins & Fonctionnalités implémentées

Exigences fonctionnelles

Exigences techniques

Cadrage

Scénario

Fonctionnalités

Cadrage fonctionnel :



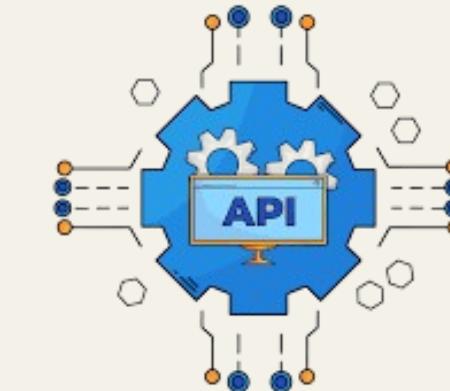
Application interne
destinée aux salariés
de DIGICHEESE



Système centré sur le
rôle Administrateur



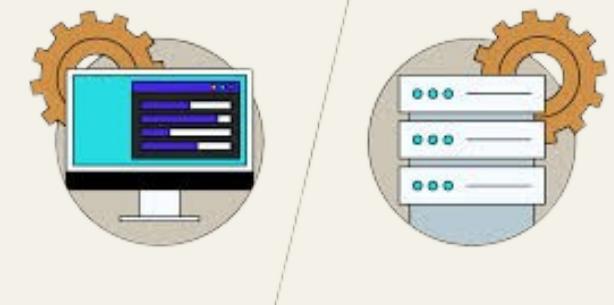
L'administrateur gère
l'ensemble des
données métier



Les opérations sont
réalisées via une API
backend



Les règles métier sont
appliquées côté
serveur



Le backend constitue
un socle pour un futur
front-end



2. Analyse des besoins & Fonctionnalités implémentées

Exigences fonctionnelles

Exigences techniques

Cadrage

Scénario

Fonctionnalités

Scénario nominal :

1) Entrée

Acteur principal :
Administrateur

- Déclencher une action CRUD
- Envoie une requête à l'API

2) Validation

API

- Vérification des droits (rôle Admin)
- Validation des formats de données
- Contrôle des règles métier

3) Traitement

Service DIGICHEESE

- Exécution de la logique métier
- Persistance des données en base

4) Sortie

API

- Retour d'une réponse JSON
- Traces et logs générés



2. Analyse des besoins & Fonctionnalités implémentées

Exigences fonctionnelles

Exigences techniques

Cadrage

Scénario

Fonctionnalités

Fonctionnalités implémentées :

Entité gérée	Fonctionnalité	Développeur
Utilisateur, Clients, Roles	CRUD	Stanislas DELANNOY
Poids, Poids-Vignettes	CRUD	Imen KHAMMASSI
Objets, Conditionnement	CRUD	Xavier DEGUERCY
Communes, Adresses	CRUD	Thi Thu Hien NGUYEN





Plan

1. Contexte, Problématique métier & Objectif du projet
2. Analyse des besoins & Fonctionnalités implémentées
- 3. Organisation, pilotage & Priorisation MoSCoW**
4. Architecture technique du backend & Modélisation logique des données
5. API : Structure & Endpoints & Gestion des règles métier et des erreurs
6. Tests automatisés et qualité logicielle
7. Outils et technologies utilisées, mise en place, exécution du projet & Résultats obtenus
8. Limites & Perspectives
9. Conclusion





Plan

3. Organisation, pilotage & Priorisation MoSCoW

3.1. Organisation & Pilotage

3.2. Priorisation MoSCoW



3. Organisation, pilotage & Priorisation MoSCoW

Organisation & Pilotage

Priorisation MoSCoW



Organisation & pilotage

- **Repo GitHub : workflow PR + intégration sur dev**
- **Backlog MoSCoW + Definition of Done (DoD) commun**
- **Rituels : daily + sync mi-journée + point de clôture (push/revue)**
- **Kanban : priorité MUST (socle → CRUD → tests → doc)**

Commit	Message	Date
.vscode	maj soir	5 days ago
DOC	calss objet_cond et class objet dans des fichiers séparés	5 days ago
UML	maj readme	2 days ago
docs_cours	calss objet_cond et class objet dans des fichiers séparés	5 days ago
src	push xavier	5 days ago
tests	maj dossiers static et templates	last week
.env	reorganisation of files	last week
.env.example	merge vendredi matin daly	5 days ago
.gitattributes	004	last week
.gitignore	maj midi	last week



3. Organisation, pilotage & Priorisation MoSCoW

Organisation & Pilotage

Priorisation MoSCoW



Priorisation (MoSCoW)

MUST (Obligatoire — validation TP7)

- M0 — Socle commun & conventions (anti-dette technique)
- M1 — Socle projet : FastAPI + MySQL + .env + démarrage (run local)
- M2 — Swagger opérationnel : endpoints documentés + exécution via Swagger (front simulé)
- 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
- M9 — Tests minimum sur CRUD Admin : au moins “happy path” + 1 cas d’erreur critique / ressource
- M10 — Documentation : DOC/ + TECH/ + UTILISATION/ (installation + lancement + tests + démo)

SHOULD (Fortement recommandé — crédibilise le projet)

- S1 — Healthcheck + version : /health + info version/build (pour diagnostic rapide)
- S2 — Qualité / conventions : ruff + règles de nommage + structure projet cohérente
- S3 — Reproductibilité : Docker + docker-compose (API + DB) pour “ça tourne chez toi / chez moi”

COULD (Option — bonus si temps)

- C1 — CRUD Opérateur Colis (client léger) : endpoints orientés usage “OP-colis”
- C2 — Sécurité simple : auth basique + rôles (Admin / OP)
- C3 — Monitoring simple : KPIs définis (même si non implémentés) + règles d’alerte documentées

WONT (Hors scope TP7 — à garder en “évolutions”)

- IHM / Front web complet (écrans, UX, etc.)
- Gestion complète des commandes (calculs, mailing, stats, impression, historisation colis, etc.)
- Module complet “OP-stocks” (gestion stock avancée)
- Process “mise à jour annuelle des stocks” + impressions associées



Plan

1. Contexte, Problématique métier & Objectif du projet
2. Analyse des besoins & Fonctionnalités implémentées
3. Organisation, pilotage & Priorisation MoSCoW
- 4. Architecture technique du backend & Modélisation logique des données**
5. API : Structure & Endpoints & Gestion des règles métier et des erreurs
6. Tests automatisés et qualité logicielle
7. Outils et technologies utilisées, mise en place, exécution du projet & Résultats obtenus
8. Limites & Perspectives
9. Conclusion





Plan

4. Architecture technique du backend & Modélisation logique des données

4.1. M0-Socle commun & Conventions

4.2. Architecture technique

4.3. Zoom Code

4.4. Modélisation logique des données



4. Architecture technique du backend & Modélisation logique des données

M0-Socle commun & Conventions

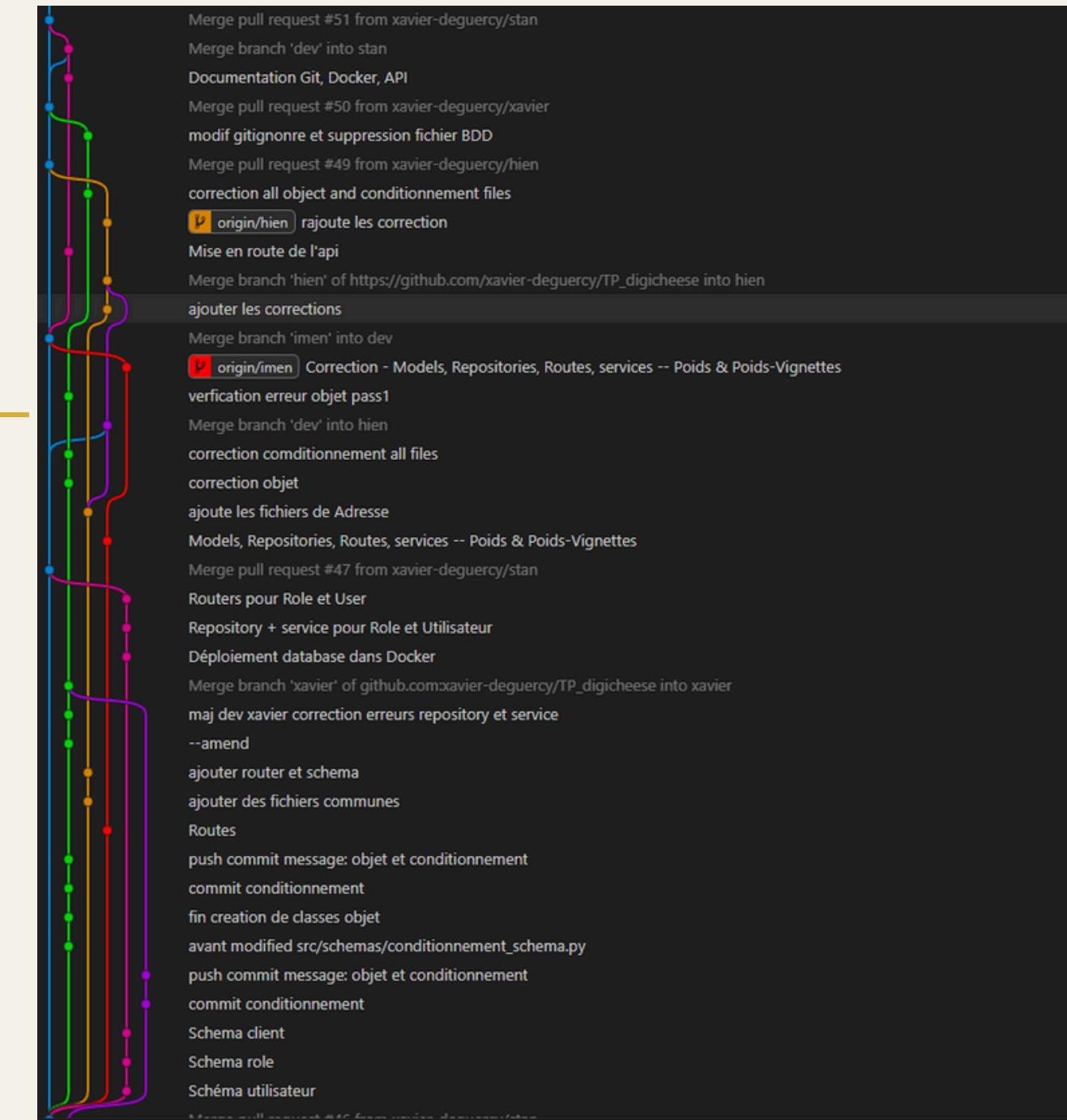
Architecture

Code

Modélisation logique des données

M0 : Socle commun & conventions (anti-dette technique)

- Conventions API : patterns CRUD, codes HTTP, erreurs standardisées
- Architecture simple : routers → services → repositories → models
- Git : feature branch → PR vers dev → rebase
 - Branches : main / prod / dev / test + 1 branche/dev
- Docker local : MySQL + phpMyAdmin + reset contrôlé



4. Architecture technique du backend & Modélisation logique des données

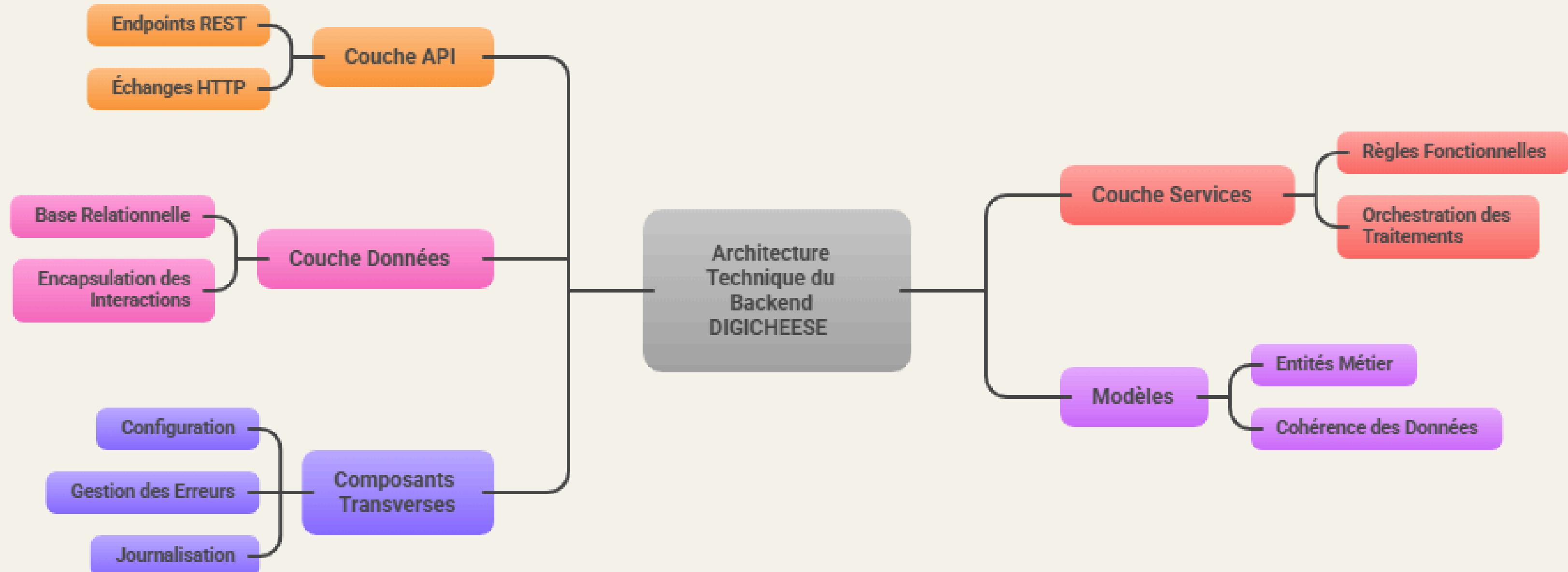
M0-Socle commun & Conventions

Architecture

Code

Modélisation logique des données

Architecture technique du Backend DIGICHEESE



4. Architecture technique du backend & Modélisation logique des données

M0-Socle commun & Conventions

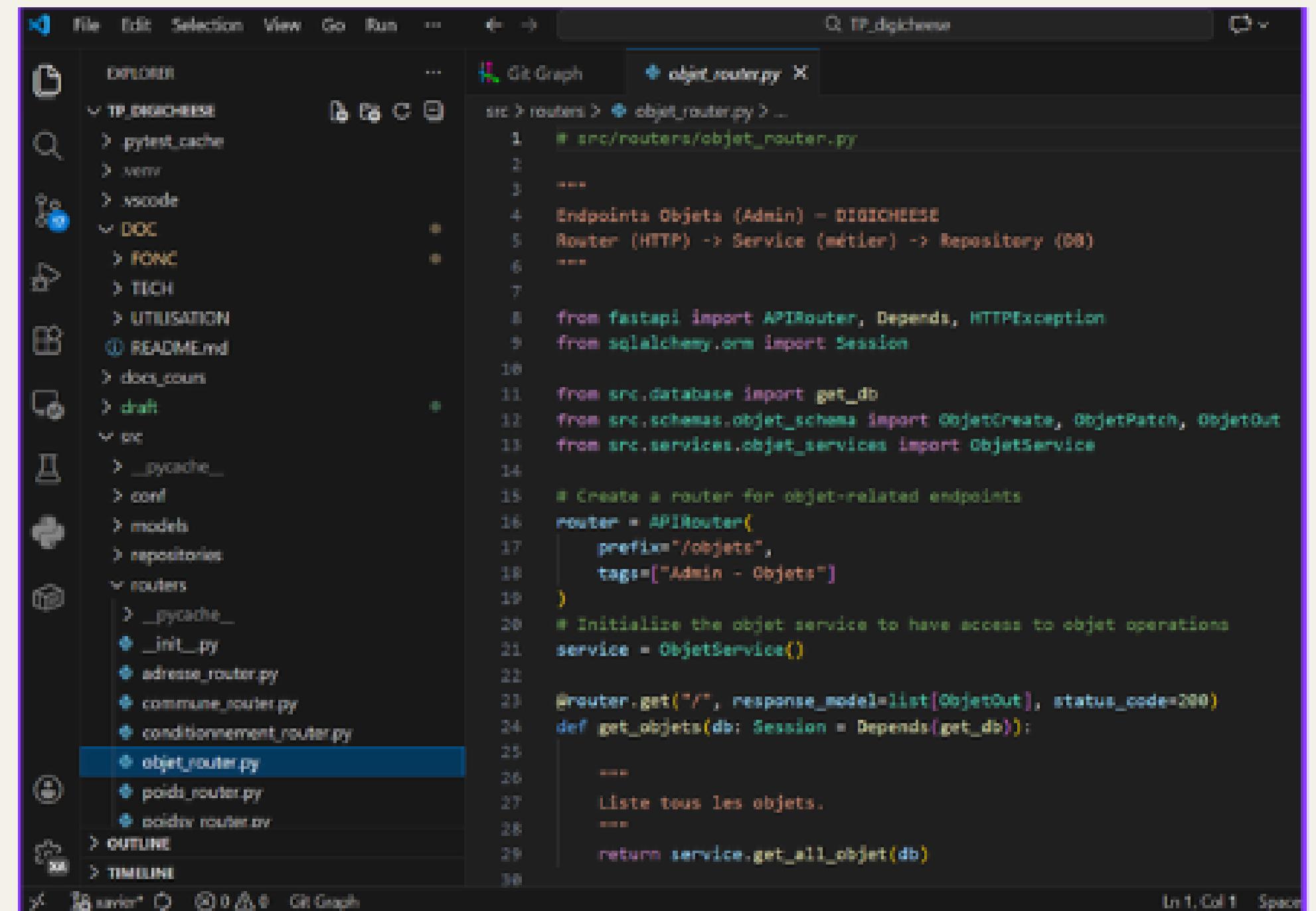
Architecture

Code

Modélisation logique des données

Zoom code :

- **Router** : définition des endpoints API et des codes HTTP.
- **Services** : règles métiers et validations.
- **Repository** : Accès et persistance des données en base.
- **Schema** : Contrats des requêtes/réponses et documentation Swagger



```
src > routers > * objet_router.py < ...
src > routers > + objet_router.py > ...
1 #!/usr/bin/env python3
2
3 # Endpoints Objets (Admin) - DISSOCHEESE
4 Router (HTTP) -> Service (métier) -> Repository (BD)
5
6
7 from fastapi import APIRouter, Depends, HTTPException
8 from sqlalchemy.orm import Session
9
10 from src.database import get_db
11 from src.schemas.objet_schemas import ObjetCreate, ObjetPatch, ObjetOut
12 from src.services.objet_services import ObjetService
13
14 # Create a router for objet-related endpoints
15 router = APIRouter(
16     prefix="/objets",
17     tags=["Admin - Objets"]
18 )
19
20 # Initialize the objet service to have access to objet operations
21 service = ObjetService()
22
23 @router.get("/", response_model=list[ObjetOut], status_code=200)
24 def get_objets(db: Session = Depends(get_db)):
25
26     """
27     Liste tous les objets.
28     """
29
30     return service.get_all_objet(db)
```

Entité	Role dans le système	Relation principale
Utilisateur	Utilisateurs du système	Associé à un ou plusieurs roles
Role	Droits et permissions	Liés aux utilisateurs
Client	Clients de DIGICHEESE	Rattaché à une commune
Commune	Référentiel géographique	Utilisée par les clients
Objet (Produit)	Articles vendus	Associé à des conditionnements
Conditionnement	Format / Emballage du produit	Lié aux objets
Poids	Seuils de calcul	Utilisé pour les frais
Poids vignette	Timbres associés au poids	Liés aux seuils de poids



Plan

1. Contexte, Problématique métier & Objectif du projet
2. Analyse des besoins & Fonctionnalités implémentées
3. Organisation, pilotage & Priorisation MoSCoW
4. Architecture technique du backend & Modélisation logique des données
5. API : Structure & Endpoints & Gestion des règles métier et des erreurs
6. Tests automatisés et qualité logicielle
7. Outils et technologies utilisées, mise en place, exécution du projet & Résultats obtenus
8. Limites & Perspectives
9. Conclusion





Plan

5. API : Structure & Endpoints & Gestion des règles métier et des erreurs

5.1. API : Structure générale

5.2. Gestion des règles métier

5.3. Gestion des erreurs



5. API : Structure & Endpoints & Gestion des règles métier et des erreurs



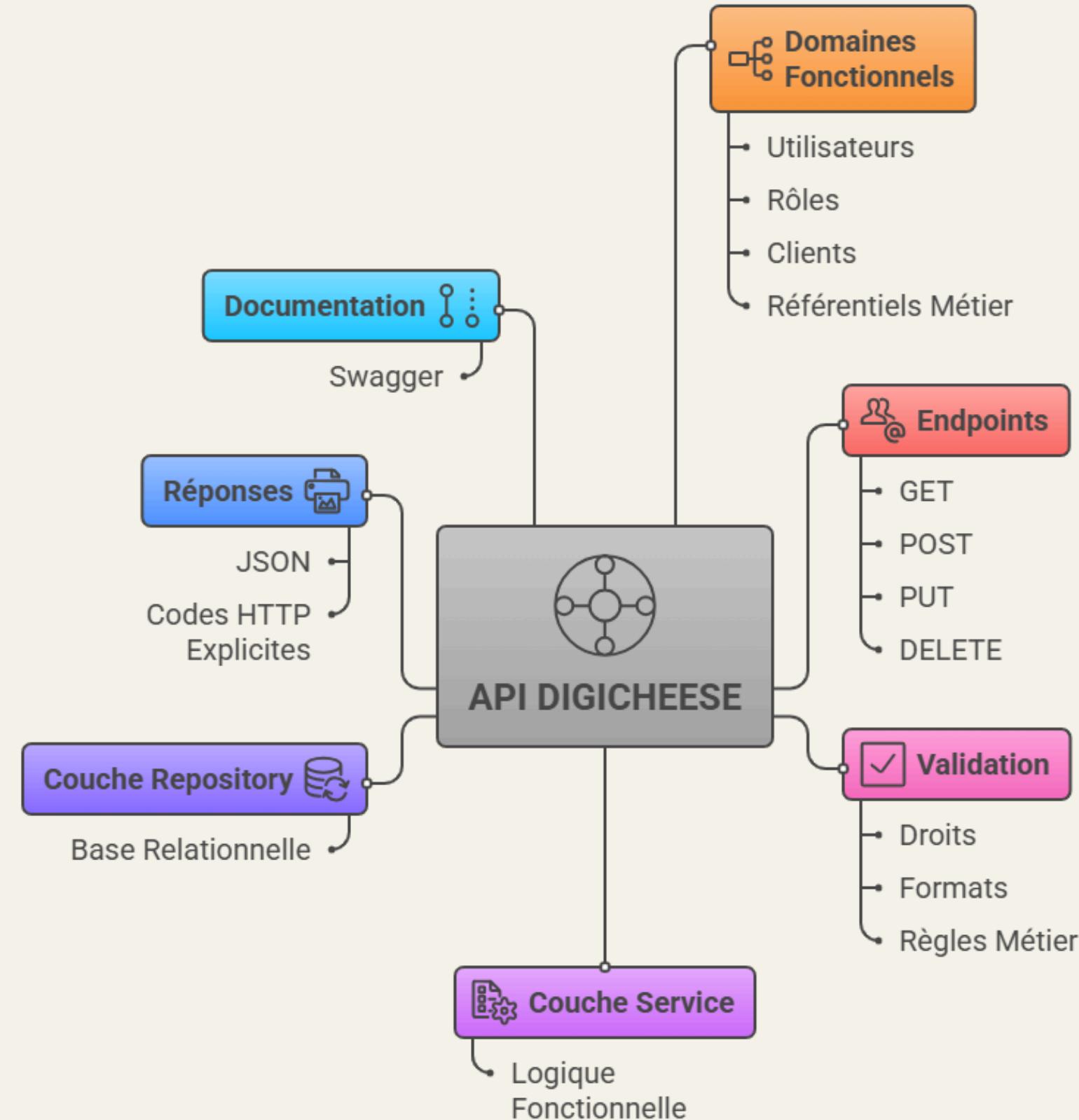
API : Structure générale

Gestion des règles métier

Gestion des erreurs

Structure d'une API DIGICHEESE

- API organisée par domaines fonctionnels.
- Endpoints REST (GET, POST, PUT/ PATCH, DELETE)
- Validation des droits et règles métiers.
- Logique métier dans la couche service.
- Accès aux données via Repository.
- Réponses JSON avec codes HTTP.
- API documentée avec Swagger.



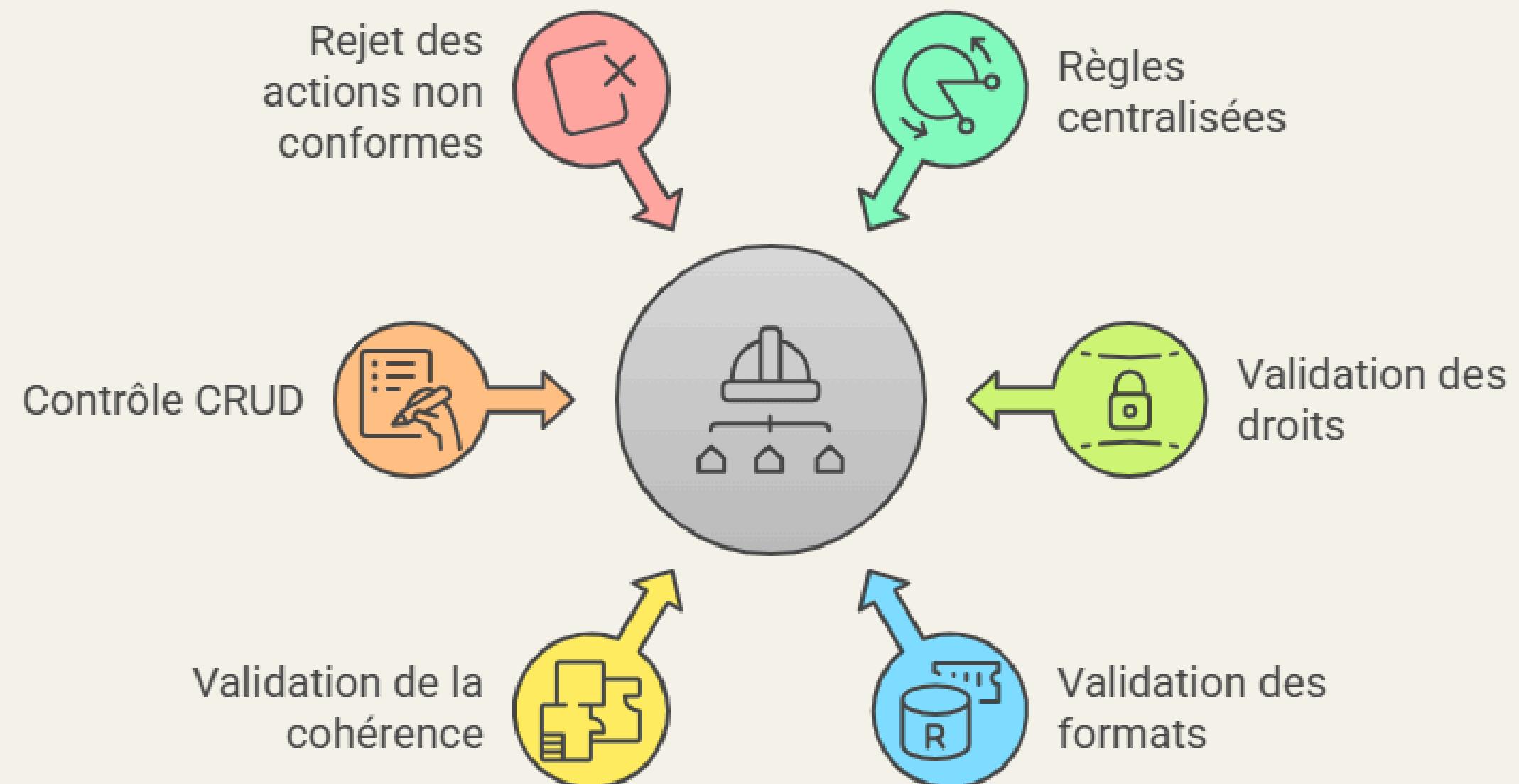
API : Structure générale

Gestion des règles métier

Gestion des erreurs

Gestion des règles métier

- Règles métier centralisées dans la couche Service
- Règles métier centralisées dans la couche Service
- Validation des droits, des formats et de la cohérence des données
- Contrôle des opérations CRUD avant persistance
- Garantit la fiabilité et la cohérence du système



API : Structure générale

Gestion des règles métier

Gestion des erreurs



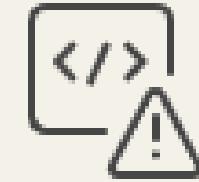
Gestion des erreurs

Logs générés



Logs générés pour le suivi et le diagnostic.

Codes HTTP

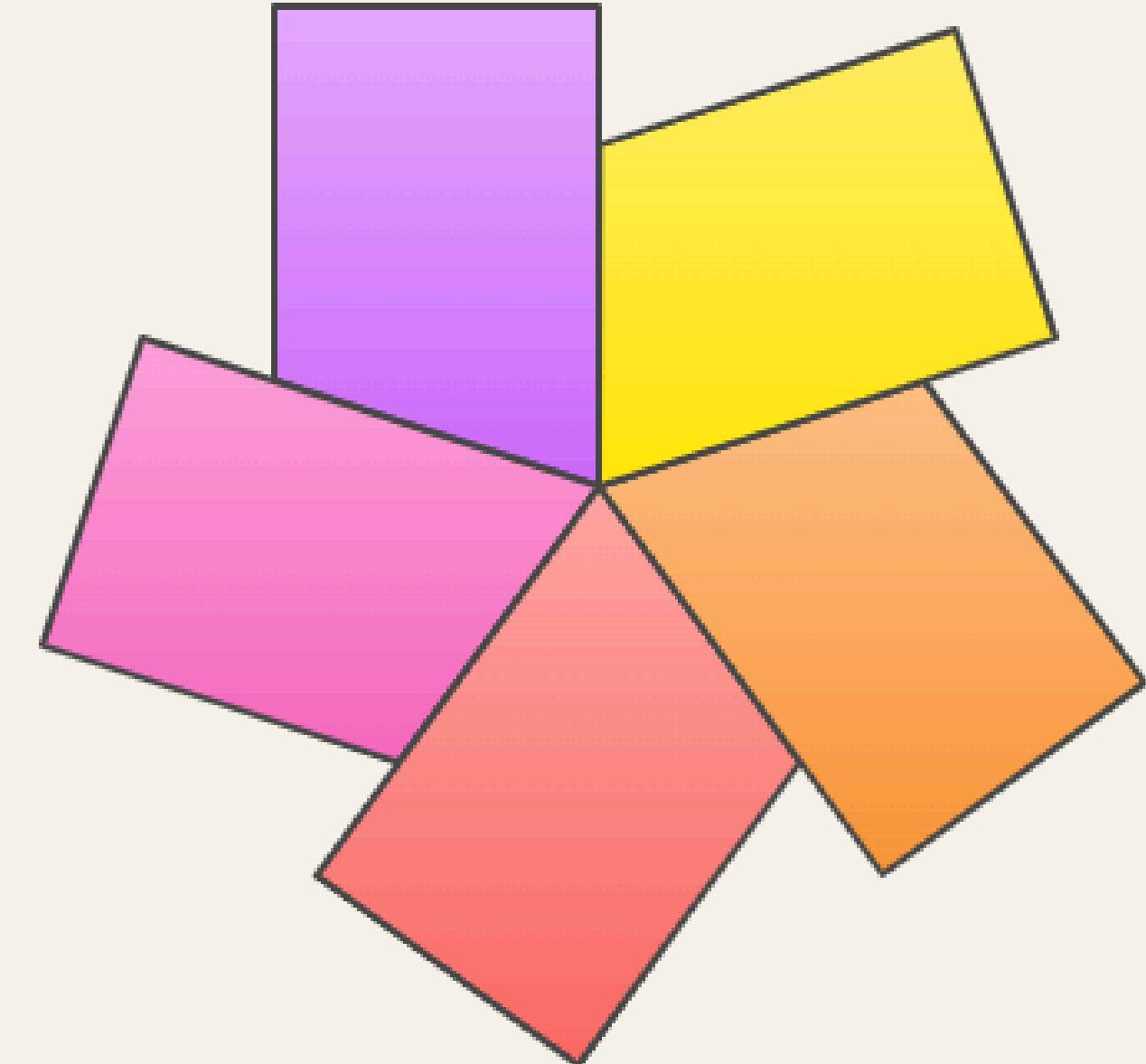


Utilisation de codes HTTP explicites (400, 404, 500...).

Retours standardisés



Retours d'erreurs standardisés en JSON.



Gestion centralisée

Gestion centralisée des erreurs dans l'API.



Vérification des erreurs

Vérification des erreurs de validation et métier.





Plan

1. Contexte, Problématique métier & Objectif du projet
2. Analyse des besoins & Fonctionnalités implémentées
3. Organisation, pilotage & Priorisation MoSCoW
4. Architecture technique du backend & Modélisation logique des données
5. API : Structure & Endpoints & Gestion des règles métier et des erreurs
6. Tests automatisés et qualité logicielle
7. Outils et technologies utilisées, mise en place, exécution du projet & Résultats obtenus
8. Limites & Perspectives
9. Conclusion





Plan

6. Tests automatisés et qualité logicielle

6.1. Définition

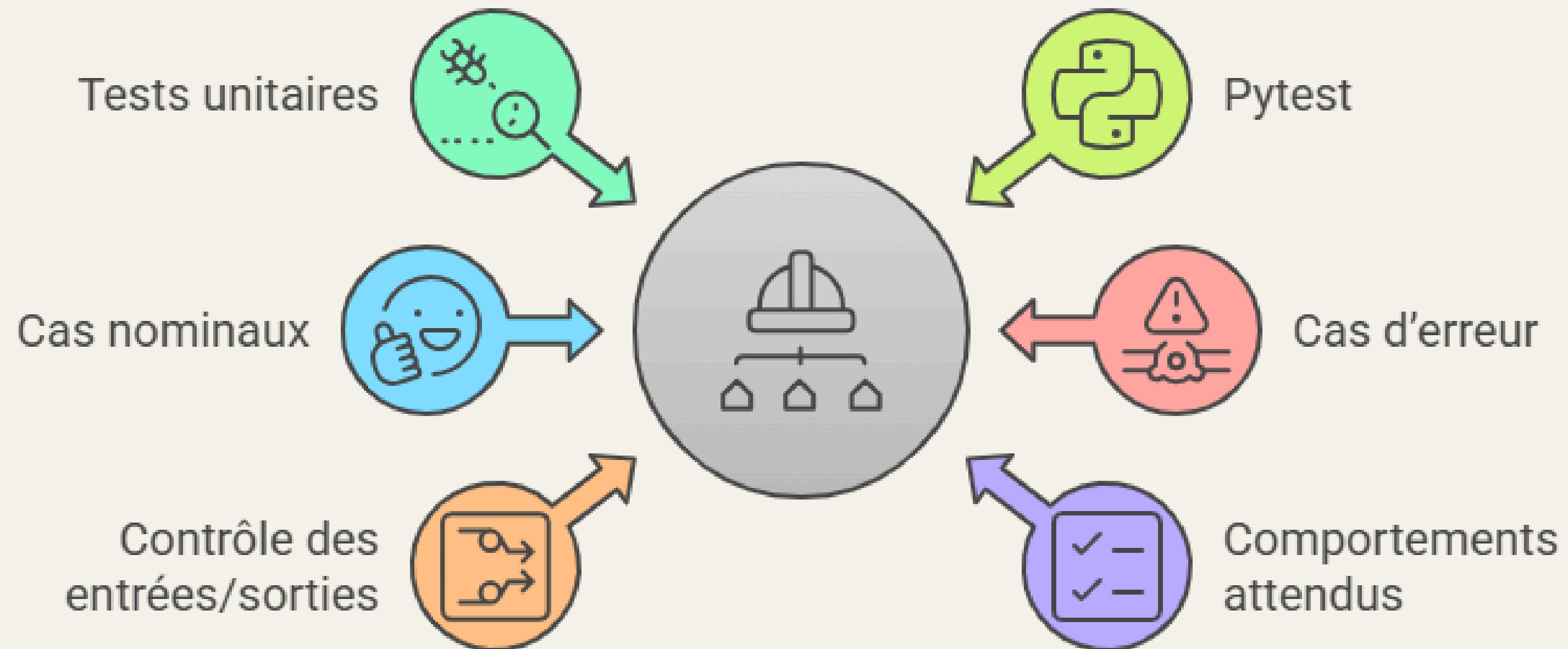
6.2. Objectifs des tests automatisés



Définition

Objectifs des tests automatisés

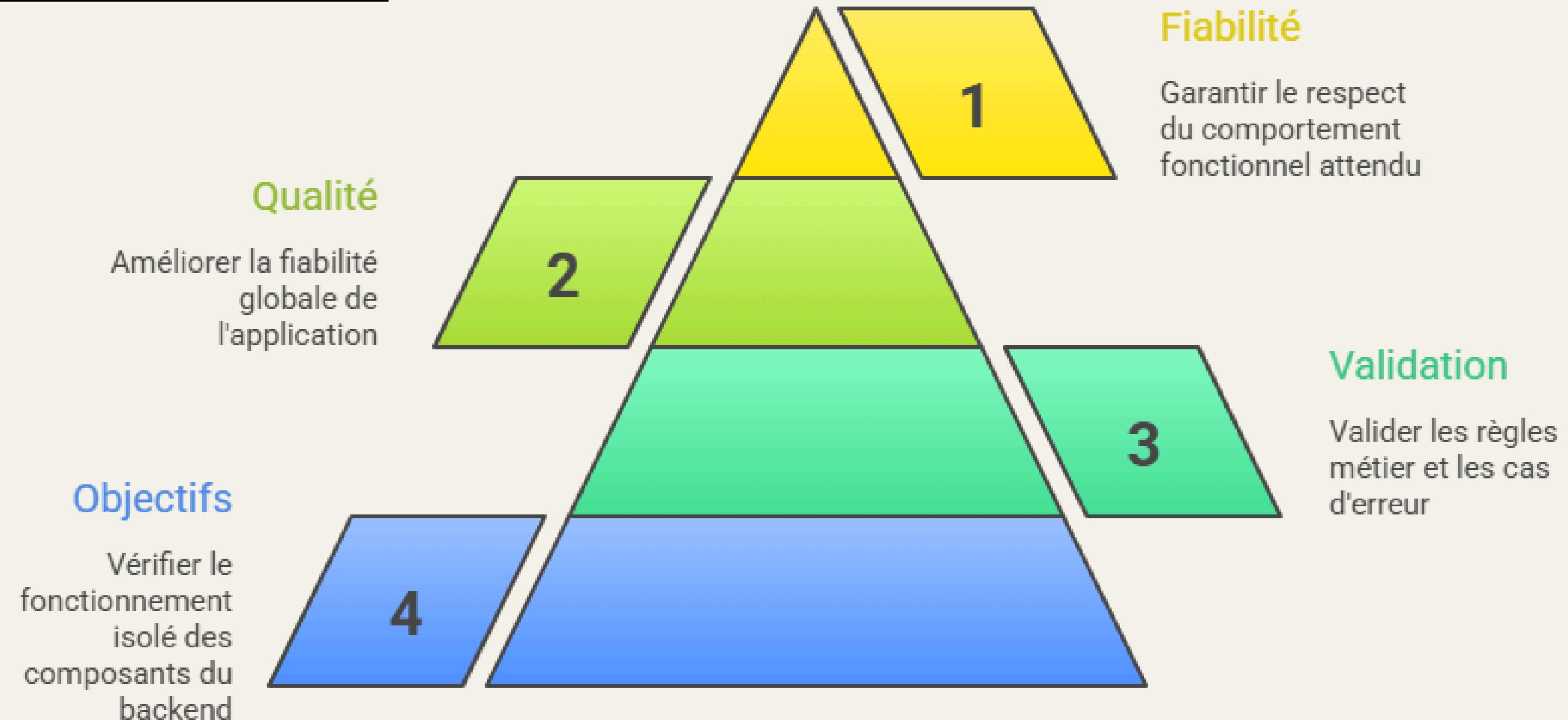
Définition des tests unitaires automatisés (PyTest):



Définition

Objectifs des tests automatisés

Pyramide des tests unitaires



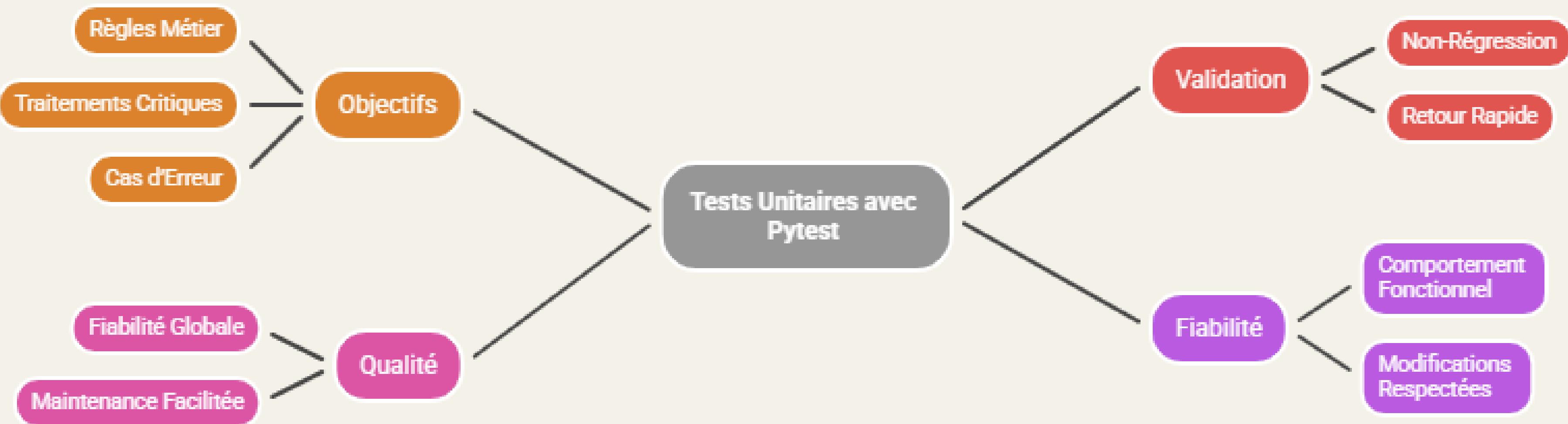
6. Tests automatisés et qualité logicielle



Définition

Objectifs des tests automatisés

Objectifs des tests unitaires avec Pytest :





Plan

1. Contexte, Problématique métier & Objectif du projet
2. Analyse des besoins & Fonctionnalités implémentées
3. Organisation, pilotage & Priorisation MoSCoW
4. Architecture technique du backend & Modélisation logique des données
5. API : Structure & Endpoints & Gestion des règles métier et des erreurs
6. Tests automatisés et qualité logicielle
7. Outils et technologies utilisées, mise en place, exécution du projet & Résultats obtenus
8. Limites & Perspectives
9. Conclusion





Plan

7. Outils et technologies utilisées, mise en place, exécution du projet & Résultats obtenus

7.1. Outils & technologies utilisées

7.3. Exécution du projet (Demo)

7.4. Résultats obtenus



7. Outils et technologies utilisées, mise en place, exécution du projet & Résultats obtenus

Outils & Technos

Exécution du projet (Demo)

Résultats obtenus



Outils & Technologies utilisées

Catégorie	Outils/ Technologies	Role dans le projet
Language	python™	Développement du backend
Framework API	FastAPI	Création de l'API REST
Documentation API	Swagger™	Documentation & Tests des endpoints
Base de données	MySQL® phpMyAdmin	Stockage des données/ Administration de la base
Tests	pytest	Tests unitaires automatisés
Versionning	git	Gestion des versions et visualisation des branches

7. Outils et technologies utilisées, mise en place, exécution du projet & Résultats obtenus

Outils & Technos

Exécution du projet (Demo)

Résultats obtenus



Plateforme		Hébergement et collaboration
Conteneurisation		Environnement de développement
IDE		Éditeur de code



7. Outils et technologies utilisées, mise en place, exécution du projet & Résultats obtenus

Technologies utilisée

Exécution du projet (Demo)

Résultats obtenus



- **Voir Demo Swagger**
- **Voir Demo des tests**



7. Outils et technologies utilisées, mise en place, exécution du projet & Résultats obtenus

Technologies utilisée

Exécution du projet (Demo)

Résultats obtenus



Résultat à J3 : CRUD livrés + socle stable

- 9 CRUD : adresse, client, commune, conditionnement, objet, poids, poids-vignette, rôle, utilisateur
- Swagger opérationnel + scénarios testables
- Environnement : docker compose + uvicorn + script create_db
- Preuves : Swagger OK + tests (happy path + erreur)

```
File Edit Selection View Go Run ... C:\TP_DIGICHEESE
TP_DIGICHEESE
models
> __init__.py
> address.py
> base.py
> client.py
> community.py
> conditionnement.py
> objet.py
> poids.py
> poidsvignette.py
> role.py
> utilisateur.py
repositories
> routers
> OUTLINE
> TIMELINE
C:\TP_DIGICHEESE
obj.py
1  from sqlalchemy import Column, Integer, String, Numeric, Boolean
2  from sqlalchemy.orm import relationship
3  from .base import Base
4
5
6  class Objet(Base):
7      """Table représentant les objets disponibles dans la fromagerie."""
8
9      __tablename__ = "t_objet" # table des objets
10
11     id_objet = Column(Integer, primary_key=True, autoincrement=True, index=True)
12     nom_objet = Column(String(50), nullable=False)
13     taille_objet = Column(String(50), nullable=True)
14
15     prix_objet = Column(Numeric(10, 4), nullable=False, default=0)
16     poids_objet = Column(Numeric(10, 4), nullable=False, default=0)
17
18     indisp_objet = Column(Boolean, nullable=False, default=False)
19     pointe_objet = Column(Integer, nullable=False, default=0)
20
21
22
23
24
25
26
27
28
29
```





Plan

1. Contexte, Problématique métier & Objectif du projet
2. Analyse des besoins & Fonctionnalités implémentées
3. Organisation, pilotage & Priorisation MoSCoW
4. Architecture technique du backend & Modélisation logique des données
5. API : Structure & Endpoints & Gestion des règles métier et des erreurs
6. Tests automatisés et qualité logicielle
7. Outils et technologies utilisées, mise en place, exécution du projet & Résultats obtenus
- 8. Limites & Perspectives**
9. Conclusion





Plan

8. Limites & Perspectives

8.1. Limites actuelles

8.2. Options & Extensions possibles

8.3. Perspectives



8. Limites & Perspectives

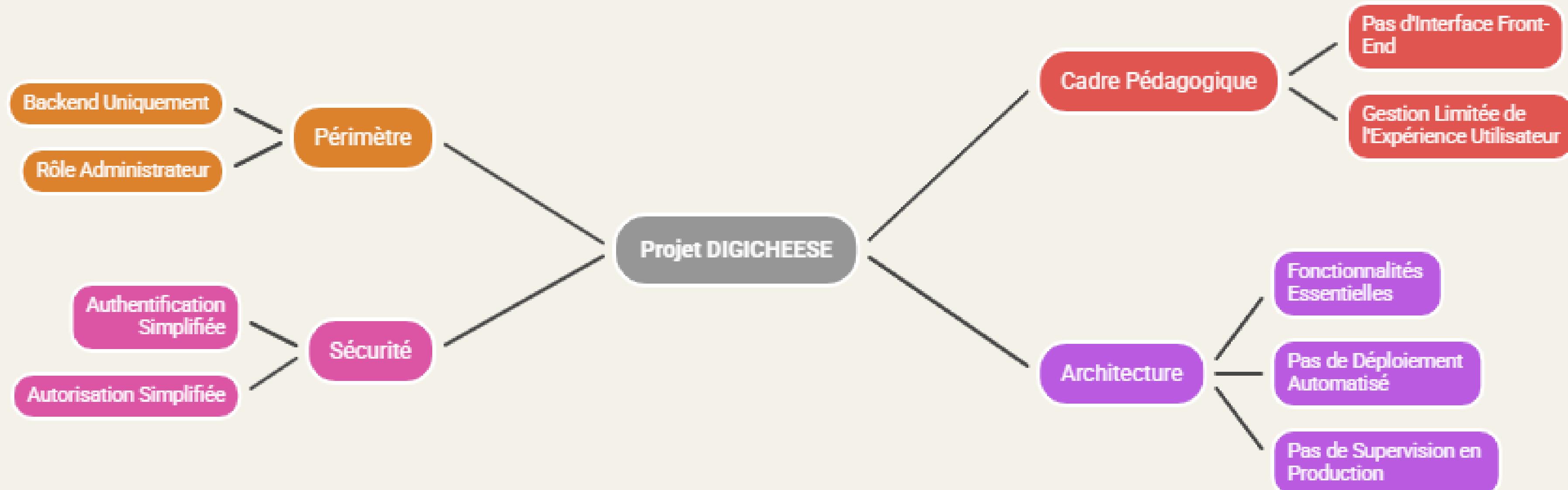
Limites actuelles

Options & Extension possibles

Perspectives



Limites actuelles du projet DIGICHEESE



8. Limites & Perspectives



Limites actuelles

Options & Extension possibles

Perspectives

Options & Extension possibles :

Opérateur Colis

Gestion des Clients

Modules Complémentaires

Fonctionnalités Métier

Projet DIGICHEESE

Aspects Techniques

Sécurité

Scalabilité

Monitoring



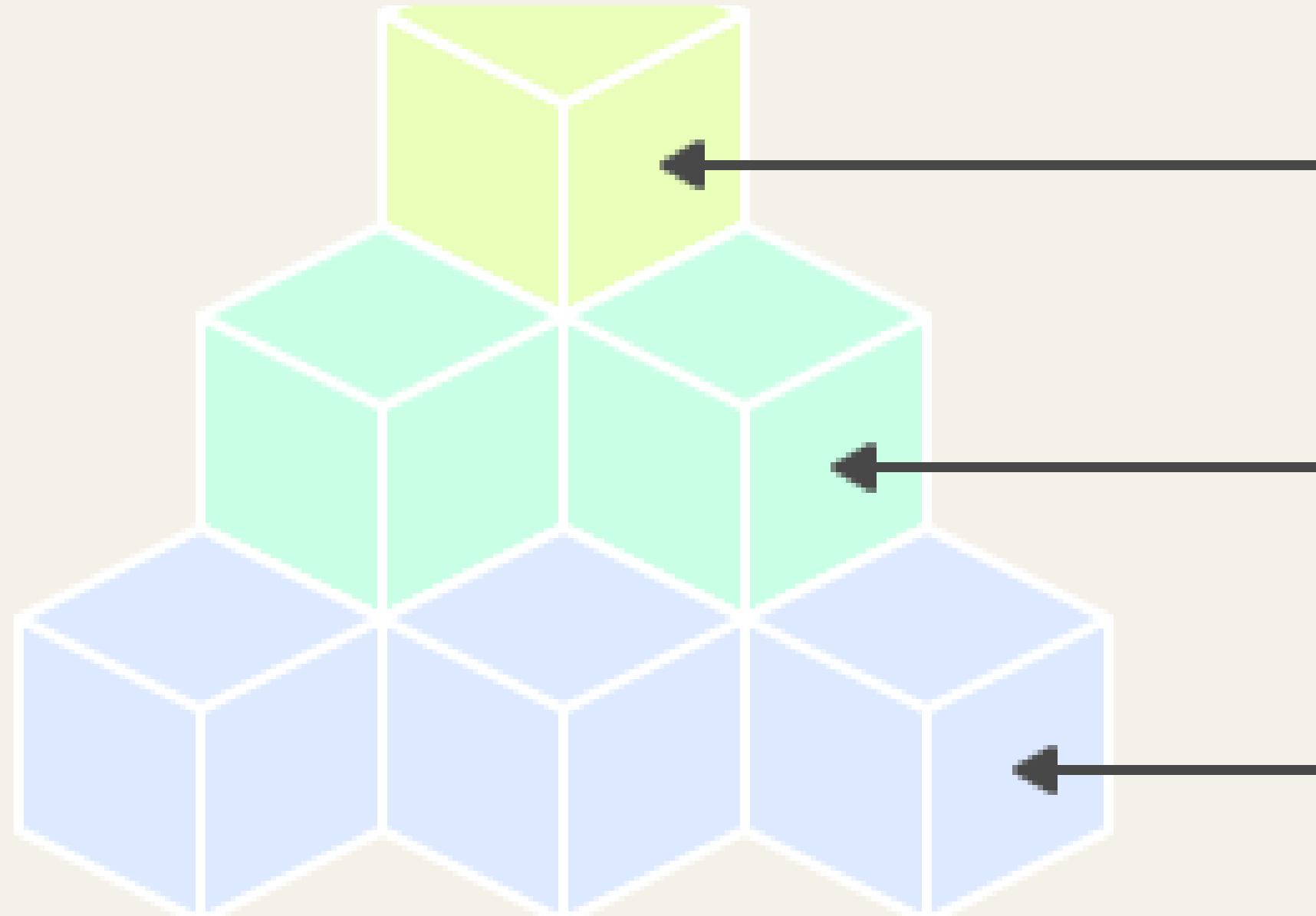
Limites actuelles

Options & Extension possibles

Perspectives



Perspectives : Pyramide d'évolution du projet DIGICHEESE



Scalabilité et automatisation

Mise en place de solutions robustes et performantes.

Interface front-end et sécurité

Amélioration de l'expérience utilisateur et de la protection des données.

Enrichissement du backend

Intégration de nouveaux rôles métiers et de fonctionnalités.



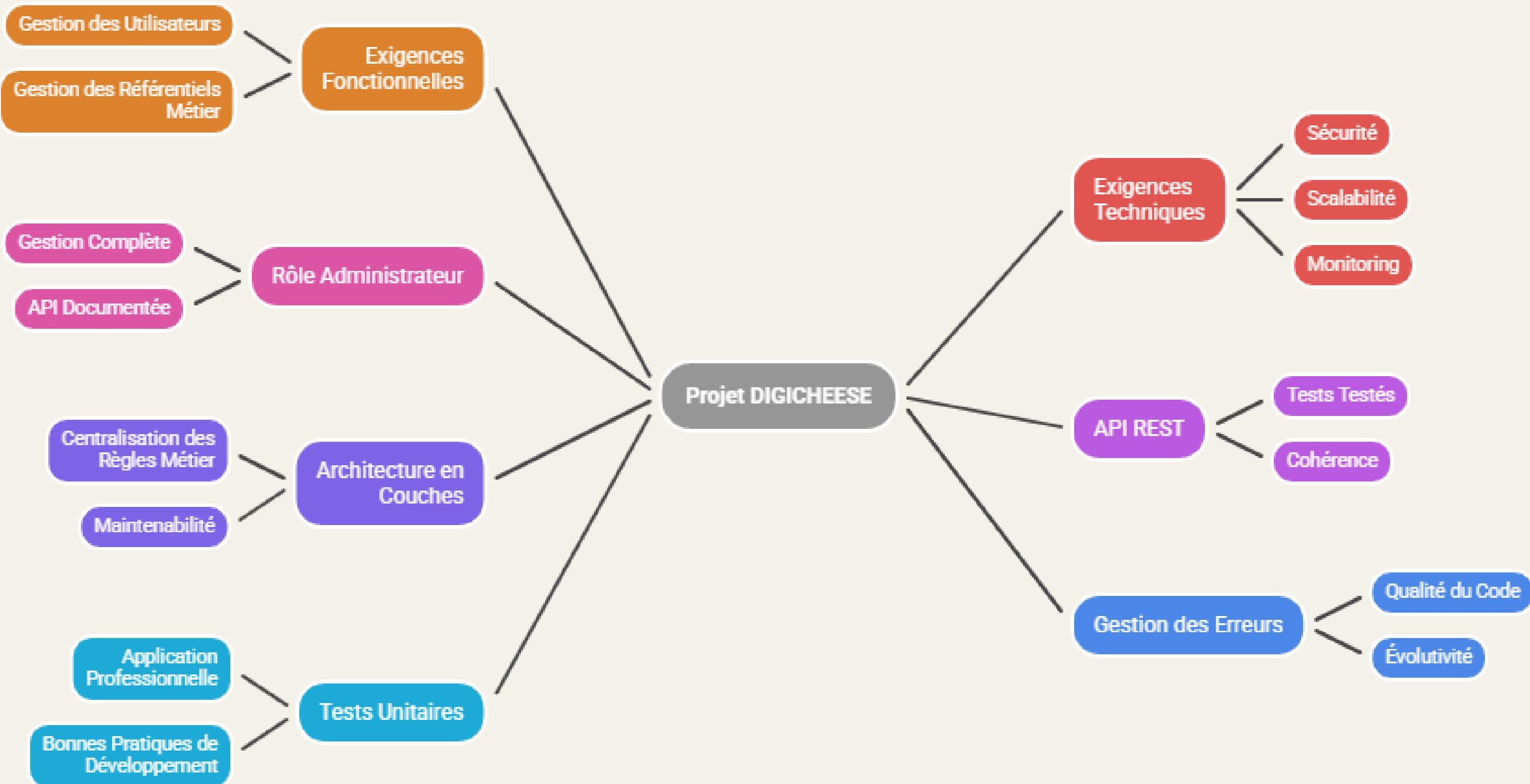
Plan

1. Contexte, Problématique métier & Objectif du projet
2. Analyse des besoins & Fonctionnalités implémentées
3. Organisation, pilotage & Priorisation MoSCoW
4. Architecture technique du backend & Modélisation logique des données
5. API : Structure & Endpoints & Gestion des règles métier et des erreurs
6. Tests automatisés et qualité logicielle
7. Outils et technologies utilisées, mise en place, exécution du projet & Résultats obtenus
8. Limites & Perspectives
9. Conclusion





9. Conclusion - Projet DIGICHEESE - Backend applicatif structuré et fonctionnel





Merci pour votre
attention !

Des questions ?



Annexe

Priorisation (MoSCoW)

Que pensez-vous du produit s'il contient **bien** cette fonctionnalité ?

Que pensez-vous du produit s'il ne contient **pas** la fonctionnalité ?

Ça me ferait plaisir C'est un minimum Je n'ai pas d'avis Je l'accepterai Ça me dérangerait

Ça me ferait plaisir	Won't have	Should have	Should have	Should have	Should have
C'est un minimum	Won't have	Could have	Could have	Could have	Must have
Je n'ai pas d'avis	Won't have	Could have	Could have	Could have	Must have
Je l'accepterai	Won't have	Could have	Could have	Could have	Must have
Ça me dérangerait	Won't have	Won't have	Won't have	Won't have	Won't have

Arborescence

```
+---DOC
| +---FONC
| +---TECH
| \---UTILISATION
+---docs_cours
+---src
| +---conf
| +---models
| | \---_pycache_
| +---repositories
| | \---_pycache_
| +---routers
| | \---_pycache_
| +---schemas
| | \---_pycache_
| +---services
| | \---_pycache_
| +---utils
| | \---_pycache_
| \---_pycache_
+---tests
```



Plan

1. Contexte, Problématique métier & Objectif du projet
2. Analyse des besoins & Fonctionnalités implémentées
3. Organisation, pilotage & Priorisation MoSCoW
4. Architecture technique du backend & Modélisation logique des données
5. API : Structure & Endpoints & Gestion des règles métier et des erreurs
6. Tests automatisés et qualité logicielle
7. Outils et technologies utilisées, mise en place, exécution du projet & Résultats obtenus
8. Limites & Perspectives
9. Conclusion

9. hien

9. hien

9. xav

9. imen

9. imen

9. 6-7 stan

9. 6-7 stan

9. xav

9. xav

