Rapport Architecture ARC42 - Système Multi-Magasins (Labs 1 et 2 - LOG430)

Auteur : Talip Koyluoglu
Date : 9 juin 2025
Projet : Système de Gestion Multi-Magasins
Repository GitHub :

- Labo 0
- Labo 1
- Labo 2

1. Introduction et Objectifs

1.1 Aperçu du Système

Ce rapport documente la transformation d'un système de caisse console simple (Lab 1) vers une application web multi-magasins (Lab 2), avec architecture modulaire et déployable via Docker.

1.2 Objectifs Architecturaux

Fonctionnalité principale du système :

- Gestion multi-magasins avec suivi des ventes et du stock.
- Fonctionnalités de réapprovisionnement manuelle.
- Consolidation des rapports de performance.
- Gestion des produits depuis la maison-mère

Contraintes techniques :

- **Scalabilité :** Ajout possible de nouveaux magasins.
- **Performance :** Réactivité aux requêtes de vente et reporting.
- **Maintenabilité :** Organisation claire en MVC.
- **Déploiement :** Conteneurisation via Docker, CI/CD GitHub Actions.

1.3 Parties Prenantes

Rôle	Responsabilités	Préoccupations
Gestionnaire	Supervision globale, apprivisionnement des magasins	Rapports stratégiques
Employé	Ventes et gestion de stock	Rapidité et fluidité de l'interface
Développeur	Maintenance, test, déploiement	Modularité, dette technique

2. Contraintes

2.1 Techniques

- Framework : Django (avec ORM intégré)
- Base de données : PostgreSQL
- Conteneurisation : Docker
- Tests : Pytest, couverture UC1 à UC6

2.2 Organisationnelles

- Livraison répartie sur 2 laboratoires (Labo 1 + Labo 2)
- Réutilisation de certaines logique du labo 1
- Déploiement reproductible

---

3. Contexte Métier

3.1 Domaine

Système de caisse et supervision multi-magasins, avec possibilité de gestion centralisée des stocks et tableaux de bord analytiques.

3.2 Évolution Lab 1 à Lab 2

- **Lab 1 :** Application console 2-tiers avec PostgreSQL.
- **Lab 2 :** Application web 3-tiers Django, MVC, Dockerisé.

---

4. Stratégie de Solution

4.1 Architecture Cible

- Architecture **3-tiers** (Frontend / Backend / DB)
- **MVC Django**
- Tests automatisés : unitaires, intégration, e2e
- Pipeline CI/CD GitHub Actions
- Données initiales via `initial_data.json`

4.2 DDD (Domain-Driven Design)

1. Domaine Central : Gestion des Ventes

- **Bounded Context :** Vente et caisse
- **Entités :** Vente, LigneVente
- **Controller :** `uc1`, `uc3`

2. Domaine de Support : Logistique

- **Bounded Context :** Stock central et local
- **Entités :** Stock, Produit
- **Controller :** `uc2`, `uc6`

3. Domaine Générique : Supervision

- **Bounded Context :** Tableaux de bord, réapprovisionnements
- **Contrôleurs :** rapports consolidés et visualisation des performances des magasins dans un tableau de bord. `uc1`, `uc3`

---

5. Vue des Composants

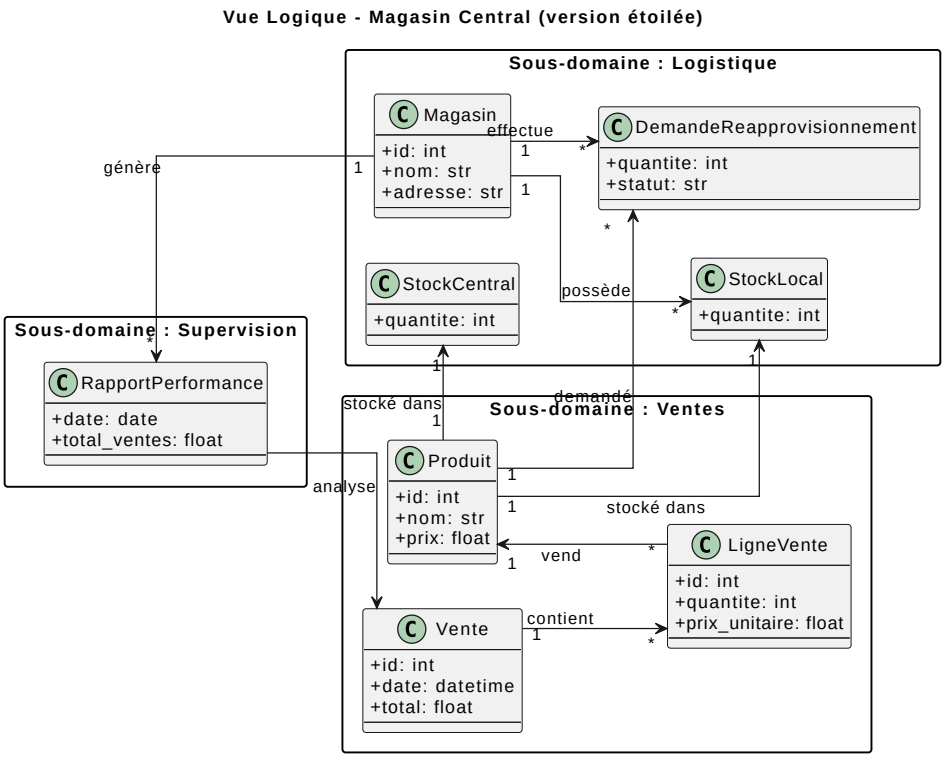
- **Frontend :** HTML + Pug templates
- **Backend :** Django views/controllers
- **Modèles :** ORM (produit, stock, vente, ligneVente)
- **Base de données :** PostgreSQL
- **Docker :** Dockerfile + docker-compose
- **CI/CD :** GitHub Actions (`ci.yml`)

---

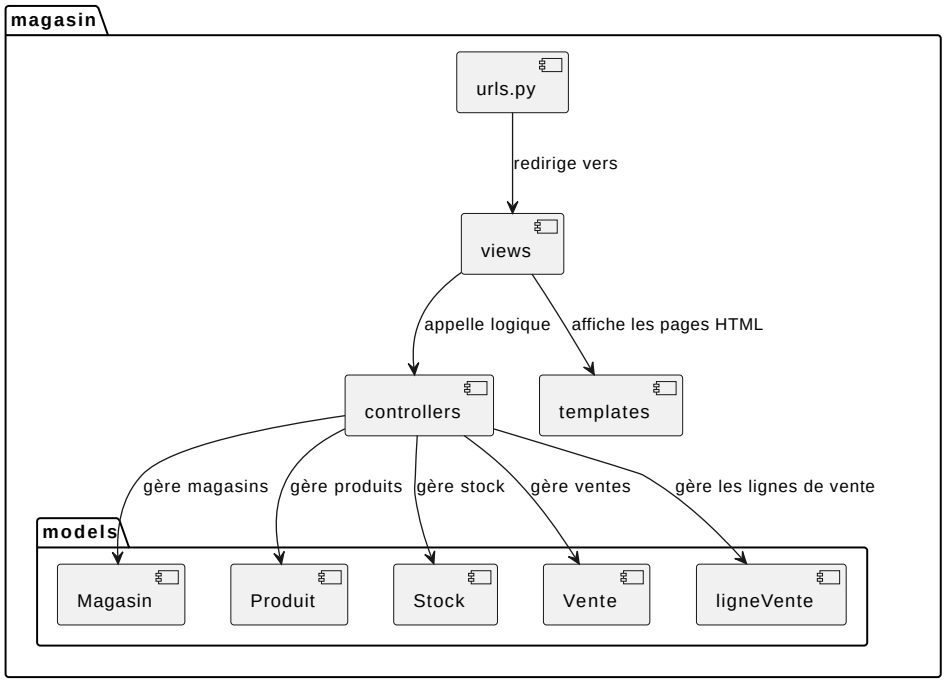
6. Vue 4+1 :

Fournie sous forme de diagrammes PlantUML :

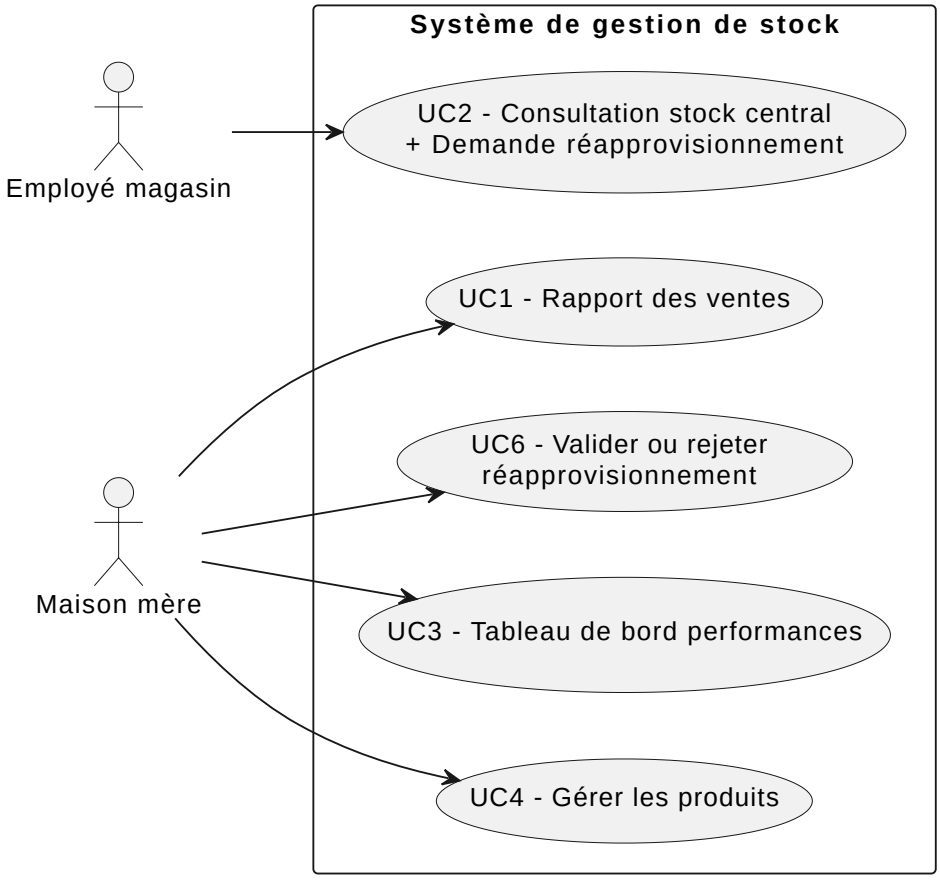
- **Voici le diagramme logique:**



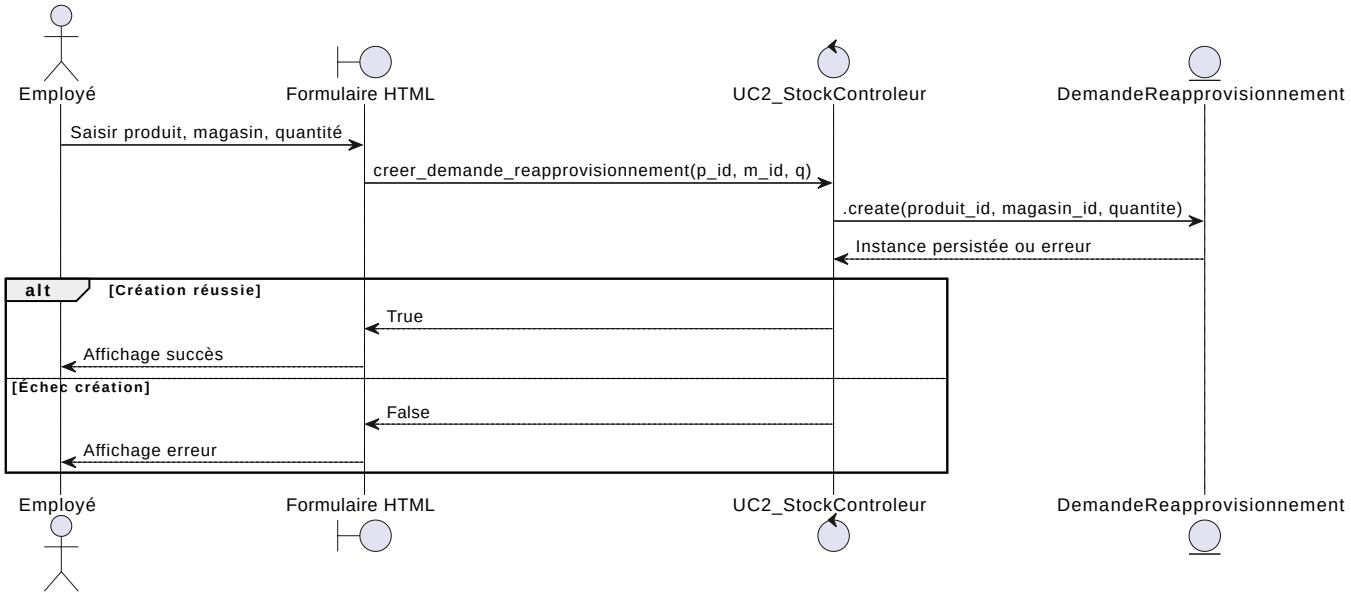
- **Voici le diagramme d'implémentation:**



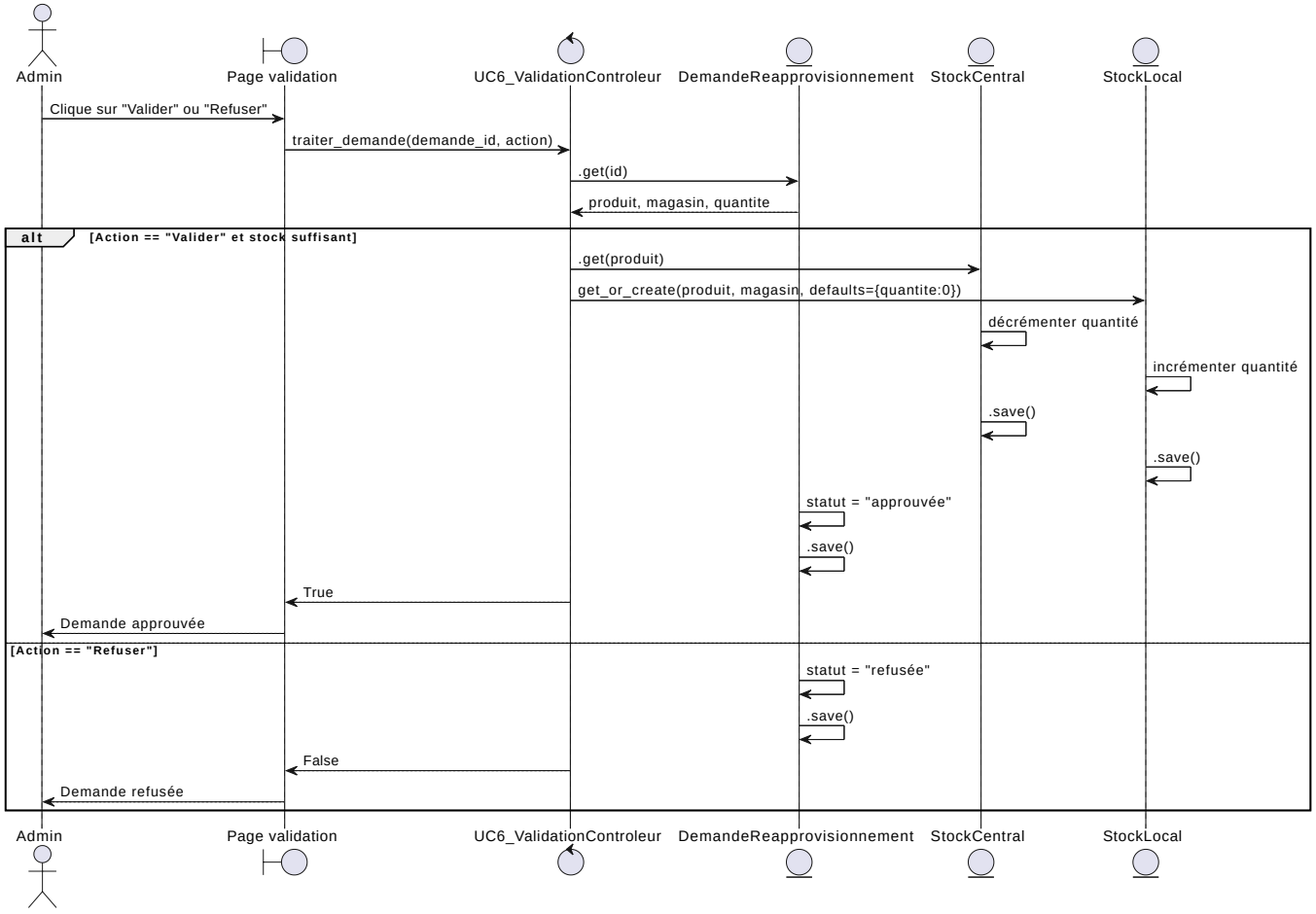
- **Voici le diagramme des cas d'utilisation:**



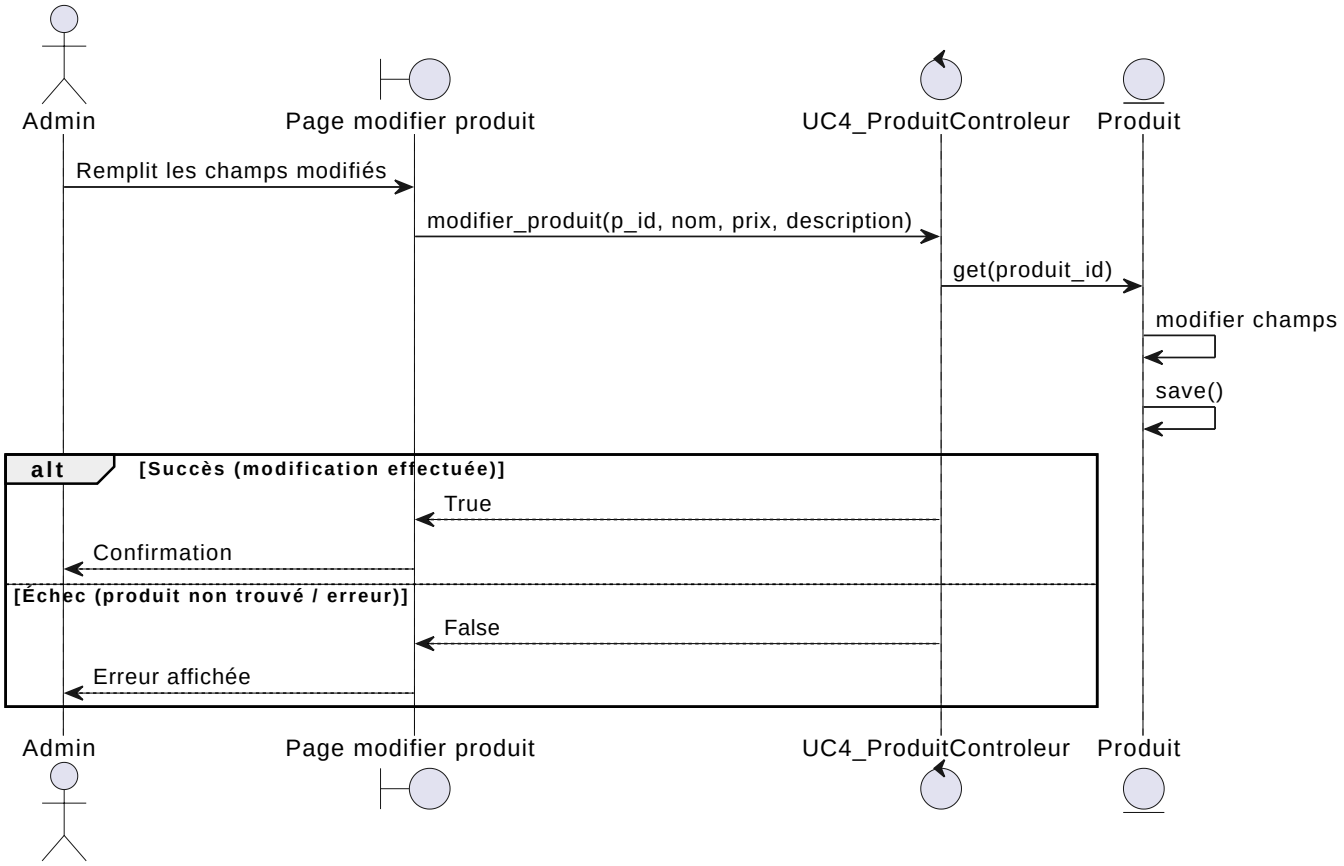
- **Voici les vues des diagrammes de séquences:**
- **DSS pour la création de demande d'apprivisionnement:**



- **DSS pour la validation d'une demande d'approvisionnement:**

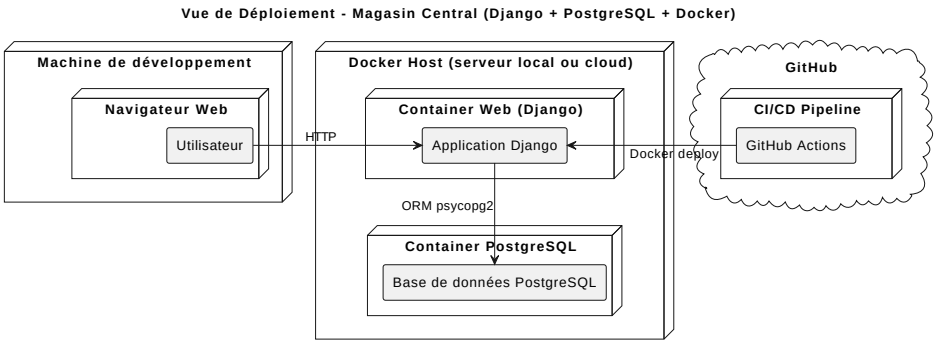


- **DSS pour la modification d'un produit depuis la maison mère:**



7. Vue de Déploiement

Voici la vue de déploiement:



---

8. Tests

- 3 classes de tests :
  - `test_unitaires.py`
  - `test_integration.py`
  - `test_e2e.py`
- Chargement automatique via `conftest.py`
- Tests fonctionnels UC1 à UC6
- Exécutés via `docker-compose run` ou dans pipeline

---

9. ADRs (Architectural Decision Records)

- **Les ADR suivant sont disponible dans le fichier /docs/ADR :
- **ADR-001 :** Adoption Architecture MVC Django
- **ADR-002 :** Adoption de l'ORM intégré de Django pour la gestion de la persistance

---

10. Risques et Dette Technique

- Risques : Absence d’authentification, surcharge requêtes
- Dette : Capacité à effectuer davantage de tests.

---

11. Glossaire

Terme	Signification
ORM	Object-Relational Mapping (modèle Django)
MVC	Model-View-Controller
CI/CD	Intégration & déploiement continu
DDD	Domain-Driven Design
UC	Use Case
3-Tier	Architecture à 3 niveaux (présentation/logique/données)
ADR	Architectural Decision Record

Conclusion

Ce système reflète une progression maîtrisée vers une architecture distribuée Django MVC, couplée à une pipeline CI/CD robuste, un usage rigoureux de DDD, et une qualité logicielle assurée par les tests et le déploiement conteneurisé.