Grille de correction détaillée – EC03

Informations générales

- **Épreuve :** EC03 – Développement back-end avec BDD relationnelle + NoSQL
- **Durée :** 4h – Individuelle
- **Projet fil rouge :** FoodSafe – Gestion des utilisateurs et profils alimentaires
- **Technologies imposées :** PostgreSQL (relationnelle) + MongoDB (NoSQL), environnement Docker fourni

Critères d'évaluation et attentes

Critère	Intitulé officiel	Attentes côté correcteur	Points de vigilance
C7.1	Environnement installé	Projet se lance via Docker (`docker-compose up`). `.env` cohérent, services PostgreSQL et MongoDB accessibles.	Vérifier logs et captures fournis. Scripts de lancement doivent fonctionner.
C8.1	POO & MVC	Code structuré (entités, contrôleurs, repositories). Respect des principes MVC.	Vérifier arborescence (`/controllers` , `/models` , `/repositories`). Noms cohérents.
C9	Sécurisation & optimisation serveur	Vérifier variables d'environnement, headers HTTP, gestion des rôles, logs sécurisés.	Aucune donnée sensible (mot de passe, token) ne doit être en clair dans le code.
C10	Base relationnelle (SQL)	Schéma PostgreSQL avec entités `User` , `AllergyProfile` .	Vérifier cohérence entre schéma, code et dump.

Critère	Intitulé officiel	Attentes côté correcteur	Points de vigilance
		Migrations ou scripts fournis. Dump SQL complet.	
C11	Base NoSQL (MongoDB)	Collections cohérentes (`Products` , `SearchHistory`). Dump JSON fourni.	Vérifier données réelles et exploitables. Pas de collections vides.
C12	Justification des choix	README explique pourquoi certaines données sont en SQL (structurées) et d'autres en NoSQL (souplesse, historique).	Justification faible = perte de points. Vérifier qualité de l'argumentaire.
C13	Sauvegarde / restauration	Scripts `backup.sh` et `restore.sh` présents et exécutables pour les deux bases.	Vérifier syntaxe des commandes. Scripts trop génériques ou inutilisables = non validé.

Livrables à corriger

- **Code source** : back-end MVC avec entités et contrôleurs
- **README.md** : explications techniques, commandes terminal, arbitrages SQL/NoSQL
- **Dumps** : SQL (`foodsafe.sql`) + JSON (`mongo.json`)
- **Scripts** : `backup.sh` , `restore.sh`
- **Captures éventuelles** : migrations, requêtes, exécution des sauvegardes/restaurations

Barème indicatif (sur 20 points)

Axe évalué	Points
Environnement Docker & terminal (C7.1)	/3
Architecture POO/MVC (C8.1)	/4

Axe évalué	Points
Sécurisation & optimisation (C9)	/2
Base relationnelle SQL (C10)	/4
Base NoSQL MongoDB (C11)	/3
Justification des choix (C12)	/2
Sauvegarde & restauration (C13)	/2

Points positifs attendus

- Code bien structuré, séparation claire MVC
- PostgreSQL utilisé pour données utilisateurs structurées (users, profils)
- MongoDB utilisé pour données dynamiques (produits, historique de recherches)
- Dumps propres, cohérents et testables
- Scripts de sauvegarde/restauration clairs et utilisables
- README complet et argumenté (arbitrages techniques SQL/NoSQL)

Erreurs fréquentes à surveiller

- MVC non respecté (contrôleur unique ou code « spaghetti »)
- Base NoSQL sous-exploitée ou absente
- Données sensibles en clair (tokens, mots de passe hardcodés)
- Dumps vides, corrompus ou incohérents avec le code
- Scripts `backup.sh` et `restore.sh` non fonctionnels ou incomplets
- README trop superficiel (« j'ai choisi Mongo parce que demandé »)

Rappel pédagogique

Cette épreuve doit confirmer que l'apprenant sait :

- Manipuler deux bases complémentaires (relationnelle + NoSQL)

- Appliquer une architecture **POO / MVC** en back-end
- Sécuriser et documenter un environnement de développement avec Docker
- Justifier des choix techniques de manière argumentée
- Automatiser la maintenance des données (sauvegarde/restauration)

Elle prépare directement l'EC04, centrée sur la **mise en place d'une API sécurisée et documentée**.