Résumé professionnel – Système de journalisation “MachineMonitor”

---

1. Contexte & problématique

- **Objectif métier** : outil interne pour enregistrer/consulter les interventions sur des machines.
- **Contraintes initiales** : fichiers JSON + UI Qt basique.
- **Besoins** :
- Fiabiliser la persistance (passage à une base relationnelle).
- Recherche/filtrage dynamique des logs.
- Exposition via une API REST (intégration web/services).

---

2. Parcours de la solution

Phase	Actions clés	Bénéfice
1. UI & JSON	Logger/Viewer Qt, sauvegardes temporaires, dialogues d’erreur.	Prototype validant le workflow sans DB.
2. SQLite & DDL	Rédaction du schéma (machines, logs), script `init_db.py`.	Centralisation, contraintes PK/FK, fiabilité.
3. sqlLib.py	Fonctions génériques CRUD (`create`, `update`, `delete`, etc.).	Abstraction, sécurité (paramétrage, rollback).
4. Migration	`syncDatabase()` pour transférer JSON → SQL.	Automatisation des imports et mises à jour.
5. API REST	FastAPI + Uvicorn, Pydantic models, routes CRUD, Swagger UI.	Validation, doc auto, tests interactifs.

---

3. Choix techniques & rôles

- **SQLite** : base embarquée ACID, introspection, légèreté.
- **DDL** : tables = onglets Excel, colonnes typées, PK/FK, NOT NULL.
- **sqlLib.py** : couche d’accès SQL réutilisable, commit/rollback.
- **FastAPI** : framework ASGI déclaratif, Swagger UI auto.
- **Uvicorn** : serveur ASGI performant.
- **Pydantic** : validation/sérialisation JSON ↔ Python.
- **cURL** : tests CLI des endpoints.

---

4. Fonctionnement global

1. `init_db.py` → création `machineMonitor.db` + tables.
2. `syncDatabase()` → JSON → SQL (INSERT/UPDATE).
3. API CRUD :

4. **GET** `/machines`, `/machines/{name}`, `/logs`, `/logs/{uuid}`
5. **POST** `/machines`, `/logs` → création
6. **PUT** `/machines/{name}`, `/logs/{uuid}` → mise à jour
7. **DELETE** … → suppression
8. Documentation interactive à `/docs`; OpenAPI en `/openapi.json`; tests via cURL.

---

5. Compétences & prochaines étapes

- **Validées** : schéma relationnel, introspection SQL, CRUD Python, API REST.
- **À venir** : joins complexes, pagination, sécurité (auth), déploiement (Docker/CI), montée en charge (PostgreSQL).