

Pour aller plus loin : persistance et mock

Persister les données avec SQLite

Nos données en mémoire disparaissent à chaque redémarrage. **SQLite** stocke une base de données complète dans un simple fichier, sans serveur à installer. Idéal pour un prototype.

Python inclut le module `sqlite3`, rien à installer :

```
import sqlite3

db = sqlite3.connect("bar.db", check_same_thread=False)
db.row_factory = sqlite3.Row    # lignes accessibles comme des dictionnaires
db.execute("""
    CREATE TABLE IF NOT EXISTS cocktails (
        id INTEGER PRIMARY KEY,
        name TEXT UNIQUE NOT NULL,
        price REAL NOT NULL,
        alcohol_free INTEGER NOT NULL DEFAULT 0
    )
""")

# Lire
rows = db.execute("SELECT * FROM cocktails WHERE alcohol_free = ?", (1,)).fetchall()
items = [dict(r) for r in rows]

# Écrire
cur = db.execute(
    "INSERT INTO cocktails (name, price, alcohol_free) VALUES (?, ?, ?)",
    (cocktail.name, cocktail.price, cocktail.alcohol_free),
)
db.commit()
```

```
new_id = cur.lastrowid
```

Toujours des **requêtes paramétrées** (les `?`), jamais de concaténation : `"... WHERE name = '" + name + "'"` est une porte ouverte à l'injection SQL. Le stagiaire de 2009 en sait quelque chose.

SQLite enregistre les booléens comme des entiers (`0` ou `1`). Pensez à les reconverter avant de les renvoyer en JSON, sinon le contrat n'est plus respecté.

Simuler une API avec Prism

En design-first, le contrat existe avant le code. [Prism](#) en tire un **faux serveur** (*mock*) qui répond selon le contrat : le front peut commencer à travailler sans attendre le back. Prism nécessite Node.js (commande `npx`).

```
npx @stoplight/prism-cli mock api-ritif.openapi.yaml
```

Le mock écoute sur <http://127.0.0.1:4010>, sans le préfixe `/v1` du serveur :

```
curl -i http://127.0.0.1:4010/cocktails/3
```

```
{"id": 3, "name": "Segfault Sunrise", "price": 9.5, "alcohol_free": false, ...}
```

- Il renvoie les `example` du contrat : de bons exemples rendent le mock utile.
- Il **valide les requêtes** : un body invalide ou une clé manquante donne la réponse d'erreur prévue par le contrat.
- Le header `Prefer` force une réponse précise, pour tester la gestion d'erreurs du client :

```
curl -i http://127.0.0.1:4010/cocktails/3 -H "Prefer: code=404"
```

```
HTTP/1.1 404 Not Found
```

```
{"type": "about:blank", "title": "Not Found", "status": 404, "detail": "Ce cocktail n'existe pas. Pas encore."}
```

Avec l'option `-d` (*dynamic*), Prism génère des données aléatoires conformes aux schémas au lieu des exemples. Pratique pour vérifier qu'un front ne casse pas avec un nom de 50 caractères.

Prism sait aussi se placer devant votre vraie API, en mode `proxy`, pour signaler chaque réponse qui ne respecte pas le contrat : `npx @stoplight/prism-cli proxy api-ritif.openapi.yaml http://localhost:8000/v1`.

Revision #3

Created 2 October 2026 20:00:54 by Thibaud FRICHET

Updated 13 October 2026 22:31:37 by Thibaud FRICHET