**1 — Qualité des données**

| **Champ** | **Type** | **Contraintes** | **Norme / Format** |
| --- | --- | --- | --- |
| user.email | string | NOT NULL, unique, format email | RFC 5322 |
| user.password | string | NOT NULL, hashé | Bcrypt |
| user.roles | JSON | NOT NULL, valide JSON | UTF-8, JSON |
| user.address | string | NOT NULL | UTF-8 |
| book.title | string | NOT NULL | UTF-8 |
| book.theme | string | NOT NULL | UTF-8 |
| book.quantity | integer | >= 0 | Entier positif |
| rental.rent\_from | date | NOT NULL | ISO 8601 (YYYY-MM-DD) |
| rental.returned | boolean | NOT NULL (0 ou 1) | booléen SQL |
| comment.rating | integer | NOT NULL, entre 1 et 5 | Valeur bornée |
| created\_at | datetime | NOT NULL | ISO 8601 (datetime) |

### 2. Fiabilité des données 1. Front-end : validation utilisateur

| **Champ** | **Validation côté navigateur** | **Exemple** |
| --- | --- | --- |
| Email | type="email" + required | Vérifie syntaxe RFC 5322 |
| Mot de passe | minlength="8" + required | Champ non vide, sécurisé |
| Champs texte | maxlength, pattern | Anti-injection, anti-spam |
| Formulaires | Bouton désactivé tant que non validé | UX + sécurité |

**2. Back-end Symfony : validation serveur**

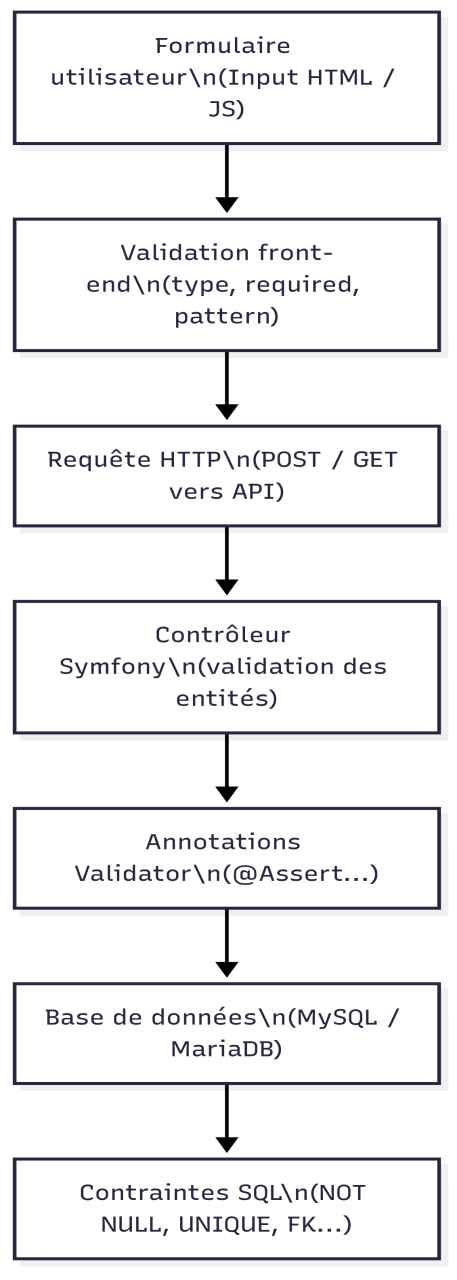
| **Entité** | **Annotation Symfony (Validator)** |
| --- | --- |
| User.email | @Assert\NotBlank, @Assert\Email |
| Book.quantity | @Assert\PositiveOrZero |
| Review.rating | @Assert\Range(min=1, max=5) |
| created\_at | @Assert\NotNull, type datetime\_immutable |

**3. Intégrité des données : base de données**

| **Mécanisme** | **Exemples dans biblioconnect.sql** |
| --- | --- |
| Clés étrangères | book\_id, user\_id, author\_id → assurent les relations |
| Contraintes | NOT NULL, UNIQUE, types int, varchar, datetime |
| Encodage | utf8mb4\_unicode\_ci → support multilingue |

**4. Sécurité des flux et du stockage**

| **Élément** | **Sécurisation** |
| --- | --- |
| Mots de passe | Hashed (bcrypt) |
| Données sensibles | Pas de stockage brut (mot de passe, rôle) |
| Accès utilisateur | Symfony Roles (ROLE\_USER, ROLE\_ADMIN) |
| Appels API / routes | Contrôle par annotations + firewall Symfony |
| Formulaires | CSRF token intégré pour prévenir les attaques |

****

**Diagramme de flux** :

3. Estimation de la charge utilisateur

Hypothèses réalistes pour BiblioConnect

| **Élément** | **Hypothèse choisie** |
| --- | --- |
| Lancement plateforme | Septembre 2025 dans une école/université |
| Utilisateurs enregistrés | 2 000 étudiants + personnel administratif |
| Connexions simultanées | 170 (aux heures de pointe : pause midi, 18h...) |
| Fonctionnalités utilisées | Connexion, recherche de livre, réservation, commentaire |

Paramètres Locust Json

{

"users\_total": 2000,

"users\_concurrent": 170,

"load\_pattern": "ramp-up: 5min -> plateau: 20min -> pic: 3min -> down: 5min"

}

Profil utilisateur typique

| **Action** | **Fréquence** | **Délai moyen entre actions** |
| --- | --- | --- |
| Connexion | 1 fois | - |
| Rechercher un livre | 3–5 fois | 3–5 secondes |
| Réserver un livre | 1 fois | après 1 recherche sur 5 |
| Consulter les avis / commentaires | 2–3 fois | 2–3 secondes |

4. Plan d’actions correctives

**Exemple réaliste pour BiblioConnect :**

| **Endpoint** | **KPI Observé** | **Seuil attendu** | **Problème identifié** | **Action Corrective** | **Délai** |
| --- | --- | --- | --- | --- | --- |
| /login | 900 ms | < 300 ms | Authentification lente | Ajouter un cache en base / opti requête | 48h |
| /book/list | 700 ms | < 500 ms | Chargement lent de tous les livres | Ajouter de la pagination / lazy loading | 72h |
| /reservation | 7% erreurs 500 | < 1 % | Validation back incomplète | Vérifier les champs requis côté back | 24h |
| /comment/add | 5% erreurs + lenteur | < 1 % et < 400ms | Trop de traitements à l’envoi | Ajouter validation front + traitement asynchrone | 72h |
| /rental/history | 1200 ms | < 500 ms | Trop de données en retour | Implémenter de la pagination + index DB | 96h |

5. Benchmark des outils JS

Comparatif :

| **Outil** | **Langage** | **Avantages** | **Inconvénients** |
| --- | --- | --- | --- |
| **k6** | JavaScript / Go | CLI rapide, scripting clair, compatible CI/CD, export vers Grafana | Peu de visualisation intégrée sans Grafana |
| **Artillery** | JavaScript | Facile à apprendre, config YAML simple, JSON output | Moins performant sur très gros volumes |
| **Jest + Puppeteer** | JavaScript | Intègre UI + API, parfait pour tests e2e + frontend | Pas adapté à de fortes charges ou scalabilité |
| **Locust** | Python | Web UI, contrôle fin, très scriptable | Pas JS natif |
| **Autocannon** | JavaScript | Hyper rapide, très bon pour Node.js / HTTP | Moins ergonomique pour des scénarios complexes |

En plus de Locust utilisé pour nos tests principaux, nous avons comparé plusieurs autres outils.  
Pour des projets en Node.js, Artillery et k6 sont particulièrement adaptés.  
k6 offre de très bonnes performances et s’intègre bien avec Grafana pour le monitoring.  
En revanche, Locust reste plus flexible pour des projets Symfony/Python comme BiblioConnect.

Liens utiles :

- [k6 Documentation](https://k6.io/docs)

- [Artillery Documentation](https://www.artillery.io/docs)

- [Puppeteer GitHub](https://github.com/puppeteer/puppeteer)

- [Locust](https://locust.io/)

- [Autocannon GitHub](https://github.com/mcollina/autocannon)

6. Résultats des tests avec Locust

Script :

from locust import HttpUser, task, between

class BiblioUser(HttpUser):

    wait\_time = between(1, 3)

    @task(2)

    def homepage(self):

        self.client.get("/", verify=False)

    @task(3)

    def list\_books(self):

        self.client.get("/book/", verify=False)

    @task(2)

    def book\_detail(self):

        self.client.get("/book/3", verify=False)

    @task(1)

    def view\_reservations(self):

        self.client.get("/reservation/", verify=False)

    @task(1)

    def login(self):

        self.client.post("/login", {

            "email": "user@biblioconnect.fr",

            "password": "password"

        }, verify=False)

    @task(1)

    def register(self):

        self.client.post("/register", {

            "email": "testuser@example.com",

            "password": "password123",

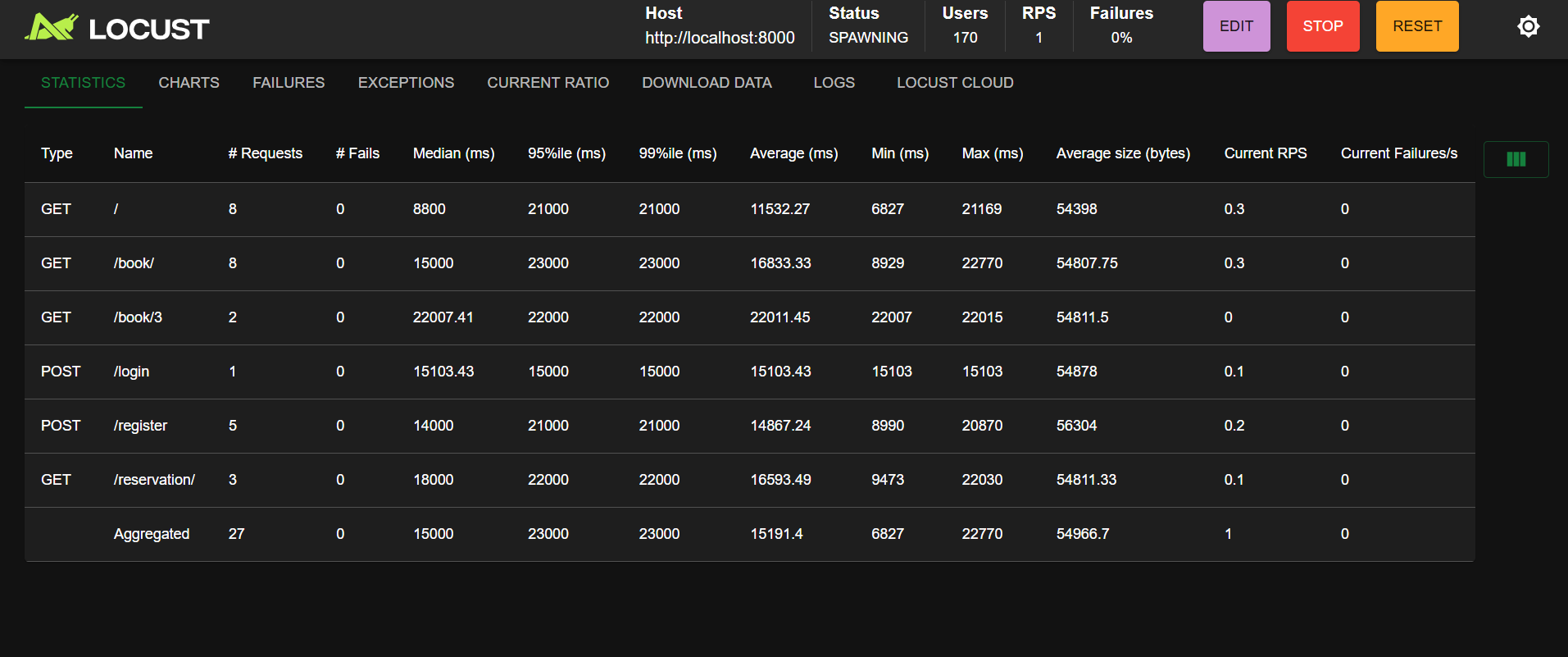
            "first\_name": "Test",

            "last\_name": "User",

            "address": "Paris"

        }, verify=False)

Commande pour lancer Locust :  
locust -f locustfile.py --host=http://localhost:8000



Le test de charge de l’application **BiblioConnect** avec 170 utilisateurs a révélé que **les routes fonctionnent sans erreurs (0 % d’échecs)**, ce qui prouve la **stabilité fonctionnelle** du backend.  
Toutefois, les **temps de réponse moyens très élevés (14s à 22s)** indiquent un besoin important d’**optimisation des performances** côté backend et base de données.  
Les endpoints /book, /reservation, et /book/[id] devront être prioritairement optimisés.  
Lors d’un second test avec une charge plus élevée, l’application a présenté plusieurs erreurs 502 – Bad Gateway sur les routes principales :

* /book/, /book/3, /login, /register, /reservation/, /

Ces erreurs indiquent que **le serveur Symfony local ne supporte pas un nombre élevé de requêtes simultanées**, probablement à cause :

* d’un manque de file d’attente,
* de surcharge CPU ou mémoire,
* ou d’un traitement trop lent (ex: absence de cache).

Cela démontre un besoin **urgent d’optimisation serveur**, comme :

* ajout d’un reverse proxy (Nginx),
* mise en cache (Redis),
* ou limitation du nombre de connexions concurrentes.

En résumé, BiblioConnect supporte une charge modérée (≈ 170 utilisateurs), mais montre ses limites à fort trafic. Les résultats obtenus justifient des actions correctives pour garantir une montée en charge stable dans un environnement réel.

7. Code source sur github

Repo LINK :

https://github.com/Dragon1ight/biblioconnect