Cahier de Charge - API Rate Limit Monitor

Objectif du Projet

Développer un système de surveillance et de gestion des limites de taux d'API (Rate Limiting) permettant de contrôler le nombre de requêtes qu'un consommateur peut effectuer dans une fenêtre de temps donnée.

Architecture Technique

Couches de l'Application

1. **Entities** - Modèles de données JPA
2. **Repositories** - Accès aux données (DAO)
3. **Services** - Logique métier
4. **Controllers** - API REST

Spécifications Détaillées

---

1. ENTITIES (Déjà implémentées)

1.1 ApiConsumer

Attributs requis:

- `id` (Long, auto-généré)
- `name` (String, obligatoire)
- `apiKey` (String, obligatoire, unique)
- `limitPerMinute` (Integer, obligatoire)
- `status` (ConsumerStatus enum, obligatoire, valeur par défaut: ACTIVE)

Contraintes:

- L'apiKey doit être unique dans la base de données
- Le nom ne peut pas être vide
- La limite par minute doit être > 0

1.2 UsageRecord

Attributs requis:

- `id` (Long, auto-généré)
- `consumer` (ManyToOne vers ApiConsumer, obligatoire)
- `windowType` (WindowType enum, obligatoire)
- `windowStart` (LocalDateTime, obligatoire)
- `requestCount` (Integer, obligatoire, valeur par défaut: 0)

Contraintes:

- Contrainte unique sur (consumer_id, window_type, window_start)
- Le nombre de requêtes ne peut pas être négatif

1.3 Enums

ConsumerStatus:

- ACTIVE - Le consommateur peut faire des requêtes
- SUSPENDED - Le consommateur est bloqué

WindowType:

- MINUTE - Fenêtre d'une minute
- HOUR - Fenêtre d'une heure

---

2. REPOSITORIES (À Implémenter)

2.1 ApiConsumerRepository

Interface: `ApiConsumerRepository extends JpaRepository<ApiConsumer, Long>`

Méthodes requises:

Méthode	Type Retour	Description
`findByApiKey(String apiKey)`	`Optional<ApiConsumer>`	Trouver un consommateur par sa clé API
`existsByApiKey(String apiKey)`	`boolean`	Vérifier si une clé API existe déjà
`findByApiKeyAndStatus(String apiKey, ConsumerStatus status)`	`Optional<ApiConsumer>`	Trouver un consommateur actif par clé API

Pourquoi ces méthodes?

- `findByApiKey`: Pour valider les requêtes entrantes
- `existsByApiKey`: Pour éviter les doublons lors de la génération de clés
- `findByApiKeyAndStatus`: Pour vérifier qu'un consommateur est actif avant de traiter sa requête

2.2 UsageRecordRepository

Interface: `UsageRecordRepository extends JpaRepository<UsageRecord, Long>`

Méthodes requises:

Méthode	Type Retour	Description
`findByConsumerAndWindowTypeAndWindowStart(ApiConsumer consumer, WindowType windowType, LocalDateTime windowStart)`	`Optional<UsageRecord>`	Trouver l'enregistrement d'utilisation pour une fenêtre spécifique

Pourquoi cette méthode?

- Pour récupérer ou créer l'enregistrement d'utilisation pour la fenêtre de temps actuelle

---

3. SERVICES (À Implémenter)

3.1 ApiConsumerService (Interface)

Package: `ma.ensa.backend.service`

Méthodes requises:

3.1.1 Création

```
java

ApiConsumer createConsumer(String name, Integer limitPerMinute)
```

Objectif: Créer un nouveau consommateur d'API

Spécifications:

- Générer automatiquement une clé API unique (UUID sans tirets)
- Valider que le nom n'est pas vide
- Valider que limitPerMinute > 0
- Status par défaut: ACTIVE
- Retourner le consommateur créé avec son apiKey

Exceptions à lancer:

- `IllegalArgumentException` si name est null/vide
- `IllegalArgumentException` si limitPerMinute <= 0

3.1.2 Consultation

```
java

ApiConsumer getConsumerByApiKey(String apiKey)
```

Objectif: Récupérer un consommateur par sa clé API

Spécifications:

- Rechercher dans la base de données
- Retourner le consommateur si trouvé

Exceptions à lancer:

- `IllegalArgumentException` si apiKey n'existe pas

```
java
```

```
ApiConsumer getConsumerById(Long id)
```

Objectif: Récupérer un consommateur par son ID

Spécifications:

- Rechercher dans la base de données
- Retourner le consommateur si trouvé

Exceptions à lancer:

- `IllegalArgumentException` si l'ID n'existe pas

```
java

List<ApiConsumer> getAllConsumers()
```

Objectif: Récupérer tous les consommateurs

Spécifications:

- Retourner la liste complète (peut être vide)

3.1.3 Mise à jour

```
java

ApiConsumer updateConsumer(Long id, String name, Integer limitPerMinute)
```

Objectif: Mettre à jour les informations d'un consommateur

Spécifications:

- Si name est fourni (non null/non vide), mettre à jour le nom
- Si limitPerMinute est fourni (> 0), mettre à jour la limite
- Ne PAS modifier l'apiKey
- Ne PAS modifier le status
- Retourner le consommateur mis à jour

Exceptions à lancer:

- `IllegalArgumentException` si l'ID n'existe pas

- `IllegalArgumentException` si limitPerMinute fourni <= 0

3.1.4 Gestion du statut

```java
void suspendConsumer(Long id)
```

Objectif: Suspendre un consommateur (passer status à SUSPENDED)

Spécifications:

- Trouver le consommateur par ID
- Changer status à SUSPENDED
- Sauvegarder en base

Exceptions à lancer:

- `IllegalArgumentException` si l'ID n'existe pas

```java
void activateConsumer(Long id)
```

Objectif: Activer un consommateur (passer status à ACTIVE)

Spécifications:

- Trouver le consommateur par ID
- Changer status à ACTIVE
- Sauvegarder en base

Exceptions à lancer:

- `IllegalArgumentException` si l'ID n'existe pas

3.1.5 Suppression

```java
void deleteConsumer(Long id)
```

Objectif: Supprimer un consommateur

Spécifications:

- Vérifier que le consommateur existe
- Supprimer de la base de données
- Note: Les UsageRecords associés seront aussi supprimés (cascade)

Exceptions à lancer:

- `IllegalArgumentException` si l'ID n'existe pas

3.2 ApiConsumerServiceImpl (Implémentation)

Package: `ma.ensa.backend.service.impl`

Annotations requises:

- `@Service` - Marquer comme composant Spring
- `@Transactional` - Gestion des transactions

Dépendances (injection par constructeur):

- `ApiConsumerRepository consumerRepository`

Méthode utilitaire privée:

```java
private String generateApiKey()
```

Objectif: Générer une clé API unique

Algorithme:

1. Générer un UUID aléatoire
2. Supprimer les tirets
3. Vérifier qu'elle n'existe pas déjà dans la base
4. Si existe, régénérer (boucle)
5. Retourner la clé unique

Exemple de génération:

```
UUID: 550e8400-e29b-41d4-a716-446655440000
Sans tirets: 550e8400e29b41d4a716446655440000
```

3.3 RateLimitService (Interface)

Package: `ma.ensa.backend.service`

Méthodes requises:

3.3.1 Vérification de limite

```java
boolean checkRateLimit(String apiKey)
```

Objectif: Vérifier si un consommateur peut faire une requête (SANS incrémenter le compteur)

Spécifications:

- Trouver le consommateur par apiKey
- Vérifier que status == ACTIVE
- Calculer la fenêtre de temps actuelle (windowStart)
- Récupérer l'UsageRecord pour cette fenêtre
- Comparer requestCount avec limitPerMinute
- Retourner true si requestCount < limitPerMinute, false sinon

Exceptions à lancer:

- `IllegalArgumentException` si apiKey invalide
- `IllegalStateException` si consommateur est SUSPENDED

3.3.2 Enregistrement de requête

```java
void recordRequest(String apiKey)
```

Objectif: Enregistrer qu'une requête a été faite (incrémenter le compteur)

Spécifications:

- Trouver le consommateur par apiKey
- Vérifier que status == ACTIVE
- Calculer la fenêtre de temps actuelle
- Trouver ou créer l'UsageRecord pour cette fenêtre
- Vérifier que requestCount < limitPerMinute AVANT d'incrémenter
- Incrémenter requestCount de 1
- Sauvegarder l'UsageRecord

Exceptions à lancer:

- `IllegalArgumentException` si apiKey invalide
- `IllegalStateException` si consommateur est SUSPENDED
- `IllegalStateException` si limite dépassée (requestCount >= limitPerMinute)

3.3.3 Consultation d'utilisation

```java
int getCurrentUsage(String apiKey, WindowType windowType)
```

Objectif: Obtenir le nombre de requêtes faites dans la fenêtre actuelle

Spécifications:

- Trouver le consommateur par apiKey
- Calculer la fenêtre de temps actuelle selon windowType
- Récupérer l'UsageRecord si existe
- Retourner requestCount, ou 0 si aucun enregistrement

Exceptions à lancer:

- `IllegalArgumentException` si apiKey invalide

3.4 RateLimitServiceImpl (Implémentation)

Package: `ma.ensa.backend.service.impl`

Annotations requises:

- `@Service`
- `@Transactional`

Dépendances (injection par constructeur):

- `ApiConsumerRepository consumerRepository`
- `UsageRecordRepository usageRecordRepository`

Méthodes utilitaires privées:

3.4.1 Calcul de fenêtre

```java
private LocalDateTime getWindowStart(WindowType windowType)
```

Objectif: Calculer le début de la fenêtre de temps actuelle

Algorithme:

1. Obtenir l'heure actuelle (LocalDateTime.now())
2. Selon windowType:
   - MINUTE: Tronquer aux minutes (10:30:45 → 10:30:00)
   - HOUR: Tronquer aux heures (10:30:45 → 10:00:00)
3. Retourner le LocalDateTime tronqué

Méthode Java à utiliser:

- `LocalDateTime.now().truncatedTo(ChronoUnit.MINUTES)` pour MINUTE
- `LocalDateTime.now().truncatedTo(ChronoUnit.HOURS)` pour HOUR

3.4.2 Trouver ou créer un enregistrement

```java
private UsageRecord findOrCreateUsageRecord(ApiConsumer consumer, WindowType windowType, LocalDateTime windo
```

Objectif: Récupérer l'UsageRecord existant ou en créer un nouveau

Algorithme:

1. Chercher dans la base: `usageRecordRepository.findByConsumerAndWindowTypeAndWindowStart(...)`
2. Si trouvé: retourner l'enregistrement existant
3. Si non trouvé:
    - Créer nouveau UsageRecord
    - Assigner consumer, windowType, windowStart
    - Initialiser requestCount à 0
    - Sauvegarder en base
    - Retourner le nouvel enregistrement

---

4. CONTROLLERS (À Implémenter)

4.1 ApiConsumerController

Package: `ma.ensa.backend.controller`

Annotations de classe:

- `@RestController`
- `@RequestMapping("/api/consumers")`

Dépendances (injection par constructeur):

- `ApiConsumerService consumerService`

Endpoints requis:

4.1.1 Créer un consommateur

```
POST /api/consumers
```

Request Body:

```json
{
  "name": "string",
  "limitPerMinute": integer
}
```

Response (201 Created):

```
json

{
  "id": 1,
  "name": "string",
  "apiKey": "string",
  "limitPerMinute": integer,
  "status": "ACTIVE"
}
```

Implémentation:

- Méthode: `@PostMapping`
- Extraire name et limitPerMinute du RequestBody (Map<String, Object>)
- Appeler `consumerService.createConsumer(name, limitPerMinute)`
- Retourner ResponseEntity avec status 201 CREATED

4.1.2 Lister tous les consommateurs

```
GET /api/consumers
```

Response (200 OK):

```
json

[
  {
    "id": 1,
    "name": "string",
    "apiKey": "string",
    "limitPerMinute": integer,
    "status": "ACTIVE"
  },
  ...
]
```

Implémentation:

- Méthode: `@GetMapping`
- Appeler `consumerService.getAllConsumers()`

- Retourner ResponseEntity avec status 200 OK

4.1.3 Obtenir un consommateur par ID

```
GET /api/consumers/{id}
```

Response (200 OK):

```
json

{
  "id": 1,
  "name": "string",
  "apiKey": "string",
  "limitPerMinute": integer,
  "status": "ACTIVE"
}
```

Implémentation:

- Méthode: `@GetMapping("/{id}")`
- Paramètre: `@PathVariable Long id`
- Appeler `consumerService.getConsumerById(id)`
- Retourner ResponseEntity avec status 200 OK

4.1.4 Obtenir un consommateur par API Key

```
GET /api/consumers/by-key/{apiKey}
```

Response (200 OK):

```
json

{
  "id": 1,
  "name": "string",
  "apiKey": "string",
  "limitPerMinute": integer,
  "status": "ACTIVE"
}
```

Implémentation:

- Méthode: `@GetMapping("/by-key/{apiKey}")`
- Paramètre: `@PathVariable String apiKey`
- Appeler `consumerService.getConsumerByApiKey(apiKey)`
- Retourner ResponseEntity avec status 200 OK

4.1.5 Mettre à jour un consommateur

```
PUT /api/consumers/{id}
```

Request Body:

```
json

{
  "name": "string",       // optionnel
  "limitPerMinute": integer // optionnel
}
```

Response (200 OK):

```
json

{
  "id": 1,
  "name": "string",
  "apiKey": "string",
  "limitPerMinute": integer,
  "status": "ACTIVE"
}
```

Implémentation:

- Méthode: `@PutMapping("/{id}")`
- Paramètres: `@PathVariable Long id`, `@RequestBody Map<String, Object> request`
- Extraire name et limitPerMinute du RequestBody
- Appeler `consumerService.updateConsumer(id, name, limitPerMinute)`
- Retourner ResponseEntity avec status 200 OK

4.1.6 Suspendre un consommateur

```
PATCH /api/consumers/{id}/suspend
```

Response (204 No Content): Pas de body

Implémentation:

- Méthode: `@PatchMapping("/{id}/suspend")`
- Paramètre: `@PathVariable Long id`
- Appeler `consumerService.suspendConsumer(id)`
- Retourner ResponseEntity avec status 204 NO_CONTENT

4.1.7 Activer un consommateur

```
PATCH /api/consumers/{id}/activate
```

Response (204 No Content): Pas de body

Implémentation:

- Méthode: `@PatchMapping("/{id}/activate")`
- Paramètre: `@PathVariable Long id`
- Appeler `consumerService.activateConsumer(id)`
- Retourner ResponseEntity avec status 204 NO_CONTENT

4.1.8 Supprimer un consommateur

```
DELETE /api/consumers/{id}
```

Response (204 No Content): Pas de body

Implémentation:

- Méthode: `@DeleteMapping("/{id}")`
- Paramètre: `@PathVariable Long id`
- Appeler `consumerService.deleteConsumer(id)`
- Retourner ResponseEntity avec status 204 NO_CONTENT

4.2 RateLimitController

Package: `ma.ensa.backend.controller`

Annotations de classe:

- `@RestController`
- `@RequestMapping("/api/rate-limit")`

Dépendances (injection par constructeur):

- `RateLimitService rateLimitService`

Endpoints requis:

4.2.1 Vérifier la limite

```
POST /api/rate-limit/check?apiKey={apiKey}
```

Response (200 OK) - Autorisé:

```json
{
  "allowed": true,
  "currentUsage": 47
}
```

Response (200 OK) - Non autorisé:

```json
{
  "allowed": false,
  "currentUsage": 100
}
```

Response (404 Not Found) - API Key invalide:

```json
```

```
{
  "error": "Consumer not found with API key: xxx"
}
```

Response (403 Forbidden) - Consommateur suspendu:

```
json

{
  "error": "Consumer is suspended"
}
```

Implémentation:

- Méthode: `@PostMapping("/check")`
- Paramètre: `@RequestParam String apiKey`
- Try-catch block:
  - Appeler `rateLimitService.checkRateLimit(apiKey)`
  - Appeler `rateLimitService.getCurrentUsage(apiKey, WindowType.MINUTE)`
  - Retourner Map avec "allowed" et "currentUsage"
  - Catch IllegalArgumentException → 404 NOT_FOUND
  - Catch IllegalStateException → 403 FORBIDDEN

4.2.2 Enregistrer une requête

```
POST /api/rate-limit/record?apiKey={apiKey}
```

Response (200 OK) - Succès:

```
json

{
  "success": true,
  "currentUsage": 48
}
```

Response (429 Too Many Requests) - Limite dépassée:

```
json
```

```
{
  "error": "Rate limit exceeded"
}
```

Response (404 Not Found) - API Key invalide:

```
json

{
  "error": "Consumer not found with API key: xxx"
}
```

Response (403 Forbidden) - Consommateur suspendu:

```
json

{
  "error": "Consumer is suspended"
}
```

Implémentation:

- Méthode: `@PostMapping("/record")`
- Paramètre: `@RequestParam String apiKey`
- Try-catch block:
  - Appeler `rateLimitService.recordRequest(apiKey)`
  - Appeler `rateLimitService.getCurrentUsage(apiKey, WindowType.MINUTE)`
  - Retourner Map avec "success" true et "currentUsage"
  - Catch IllegalArgumentException → 404 NOT_FOUND
  - Catch IllegalStateException (suspended) → 403 FORBIDDEN
  - Catch IllegalStateException (rate limit) → 429 TOO_MANY_REQUESTS

4.2.3 Obtenir l'utilisation actuelle

```
GET /api/rate-limit/usage?apiKey={apiKey}&windowType={MINUTE|HOUR}
```

Query Parameters:

- apiKey (requis)

- windowType (optionnel, défaut: MINUTE)

Response (200 OK):

```json
{
  "apiKey": "string",
  "windowType": "MINUTE",
  "currentUsage": 47
}
```

Response (404 Not Found):

```json
{
  "error": "Consumer not found with API key: xxx"
}
```

Implémentation:

- Méthode: `@GetMapping("/usage")`
- Paramètres:
  - `@RequestParam String apiKey`
  - `@RequestParam(defaultValue = "MINUTE") WindowType windowType`
- Try-catch block:
  - Appeler `rateLimitService.getCurrentUsage(apiKey, windowType)`
  - Retourner Map avec "apiKey", "windowType", "currentUsage"
  - Catch IllegalArgumentException → 404 NOT_FOUND

4.3 GlobalExceptionHandler

Package: `ma.ensa.backend.controller`

Annotation de classe:

- `@RestControllerAdvice`

Objectif: Gérer les exceptions de manière centralisée

Méthodes requises:

4.3.1 Gérer IllegalArgumentException

```java
@ExceptionHandler(IllegalArgumentException.class)
public ResponseEntity<Map<String, String>> handleIllegalArgument(IllegalArgumentException e)
```

Comportement:

- Retourner status 400 BAD_REQUEST
- Body: `{"error": "message de l'exception"}`

4.3.2 Gérer IllegalStateException

```java
@ExceptionHandler(IllegalStateException.class)
public ResponseEntity<Map<String, String>> handleIllegalState(IllegalStateException e)
```

Comportement:

- Retourner status 403 FORBIDDEN
- Body: `{"error": "message de l'exception"}`

4.3.3 Gérer toutes les autres exceptions

```java
@ExceptionHandler(Exception.class)
public ResponseEntity<Map<String, String>> handleGenericException(Exception e)
```

Comportement:

- Retourner status 500 INTERNAL_SERVER_ERROR
- Body: `{"error": "An unexpected error occurred"}`
- Note: Ne PAS exposer le message de l'exception pour la sécurité

---

5. TESTS FONCTIONNELS

Scénario 1: Créer et utiliser un consommateur

Étapes:

1. Créer un consommateur avec limite de 5/minute
2. Faire 5 requêtes → Toutes acceptées
3. Faire la 6ème requête → Rejetée (429)
4. Attendre 1 minute
5. Faire une requête → Acceptée (nouveau window)

Scénario 2: Suspension

Étapes:

1. Créer un consommateur
2. Faire une requête → Acceptée
3. Suspendre le consommateur
4. Faire une requête → Rejetée (403 Forbidden)
5. Activer le consommateur
6. Faire une requête → Acceptée

Scénario 3: Mise à jour de limite

Étapes:

1. Créer un consommateur avec limite de 10/minute
2. Faire 8 requêtes → Acceptées
3. Mettre à jour la limite à 5/minute
4. Faire une requête → Rejetée (déjà 8 > 5)
5. Attendre nouvelle fenêtre
6. Faire 4 requêtes → Acceptées
7. Faire la 5ème → Rejetée

---

6. CRITÈRES DE RÉUSSITE

Votre implémentation est complète si:

✅ Tous les repositories ont les bonnes méthodes
✅ Tous les services implémentent correctement les interfaces
✅ Tous les controllers exposent les bons endpoints
✅ Les exceptions sont bien gérées avec les bons codes HTTP
✅ La génération d'API Key est unique
✅ Le calcul de fenêtre de temps fonctionne correctement
✅ Les compteurs s'incrémentent correctement
✅ Les limites sont respectées
✅ Les status ACTIVE/SUSPENDED fonctionnent
✅ Les 3 scénarios de test passent

---

7. CONSEILS D'IMPLÉMENTATION

Ordre recommandé:

1. **Repositories** (le plus simple)
    - Définir les interfaces avec les bonnes méthodes
2. **ApiConsumerService** (logique simple)
    - Implémenter l'interface
    - Commencer par les méthodes CRUD basiques
    - Puis ajouter suspend/activate
3. **RateLimitService** (logique complexe)
    - Implémenter getWindowStart() d'abord
    - Puis findOrCreateUsageRecord()
    - Ensuite checkRateLimit()
    - Enfin recordRequest() et getCurrentUsage()
4. **Controllers** (le plus simple, juste mapper aux services)
    - ApiConsumerController en premier
    - Puis RateLimitController
    - Enfin GlobalExceptionHandler
5. **Tests manuels avec Postman**
    - Tester chaque endpoint un par un

- Valider les 3 scénarios

Rappels importants:

- Toujours valider les entrées
- Utiliser @Transactional sur les services
- Injection de dépendances par constructeur
- Try-catch dans les controllers pour gérer les exceptions
- Ne jamais retourner null, utiliser Optional
- Messages d'erreur clairs et explicites

Bon courage! 🚀

Commence par les repositories, puis remonte vers les controllers.
Si tu bloques sur une méthode, réfléchis d'abord à:

1. Quelle est l'entrée?
2. Quelle est la sortie attendue?
3. Quelles validations sont nécessaires?
4. Quelle exception lancer si ça échoue?