Documentation des API - Application de Réservation d'Événements

Introduction

Ce document présente l'ensemble des API REST disponibles dans l'application de réservation d'événements développée en architecture microservices. L'application se compose de trois services principaux :

- API Gateway (point d'entrée unique)
- Event Service (gestion des événements)
- Booking Service (gestion des réservations)

L'application ne dispose pas d'interface utilisateur front-end, mais toutes les fonctionnalités sont accessibles via ces API REST.

API des Événements

1. Lister tous les événements

Endpoint: `GET /api/events`

Description: Récupère la liste de tous les événements disponibles.

Exemple de requête:

```
curl -X GET http://localhost:8080/api/events
```

Exemple de réponse:

```
[
  {
    "id": 1,
    "name": "Concert de Jazz",
    "total_seats": 100,
    "available_seats": 100,
    "event_date": "2025-06-15T20:00:00.000Z",
    "created_at": "2023-05-15T10:30:00.000Z",
    "updated_at": "2023-05-15T10:30:00.000Z"
  },
  {
    "id": 2,
    "name": "Conférence Tech",
    "total_seats": 50,
    "available_seats": 50,
    "event_date": "2025-06-20T10:00:00.000Z",
    "created_at": "2023-05-15T10:30:00.000Z",
    "updated_at": "2023-05-15T10:30:00.000Z"
  },
  {
    "id": 3,
    "name": "Spectacle de Danse",
    "total_seats": 200,
    "available_seats": 200,
    "event_date": "2025-07-01T19:30:00.000Z",
    "created_at": "2023-05-15T10:30:00.000Z",
    "updated_at": "2023-05-15T10:30:00.000Z"
  }
]
```

2. Récupérer un événement spécifique

Endpoint: `GET /api/events/{id}`

Description: Récupère les détails d'un événement spécifique par son ID.

Exemple de requête:

```
curl -X GET http://localhost:8080/api/events/1
```

Exemple de réponse:

```
{
  "id": 1,
  "name": "Concert de Jazz",
  "total_seats": 100,
  "available_seats": 100,
  "event_date": "2025-06-15T20:00:00.000Z",
  "created_at": "2023-05-15T10:30:00.000Z",
  "updated_at": "2023-05-15T10:30:00.000Z"
}
```

En cas d'événement non trouvé (HTTP 404):

```
{
  "error": "Événement non trouvé"
}
```

3. Créer un nouvel événement

Endpoint: `POST /api/events`

Description: Crée un nouvel événement avec les détails fournis.

Corps de la requête:

```
{
  "name": "Concert Rock",
  "total_seats": 150,
  "event_date": "2025-08-15 21:00:00"
}
```

Exemple de requête:

```
curl -X POST http://localhost:8080/api/events \
  -H "Content-Type: application/json" \
  -d '{
    "name": "Concert Rock",
    "total_seats": 150,
    "event_date": "2025-08-15 21:00:00"
  }'
```

Exemple de réponse (HTTP 201):

```
{
  "id": 4,
  "name": "Concert Rock",
  "total_seats": 150,
  "available_seats": 150,
  "event_date": "2025-08-15T21:00:00.000Z"
}
```

En cas d'erreur de validation (HTTP 400):

```
{
  "error": "Tous les champs sont requis (name, total_seats,
event_date)"
```

```
}
```

4. Mettre à jour un événement

Endpoint: `PUT /api/events/{id}`

Description: Met à jour les détails d'un événement existant.

Corps de la requête:

```
{
  "name": "Concert de Jazz (Modifié)",
  "total_seats": 120,
  "available_seats": 120,
  "event_date": "2025-06-15 20:30:00"
}
```

Exemple de requête:

```
curl -X PUT http://localhost:8080/api/events/1 \
  -H "Content-Type: application/json" \
  -d '{
    "name": "Concert de Jazz (Modifié)",
    "total_seats": 120,
    "available_seats": 120,
    "event_date": "2025-06-15 20:30:00"
  }'
```

Exemple de réponse:

```
{
  "id": 1,
  "name": "Concert de Jazz (Modifié)",
```

```
  "total_seats": 120,
  "available_seats": 120,
  "event_date": "2025-06-15T20:30:00.000Z"
}
```

En cas d'événement non trouvé (HTTP 404):

```
{
  "error": "Événement non trouvé"
}
```

5. Supprimer un événement

Endpoint: `DELETE /api/events/{id}`

Description: Supprime un événement spécifique et toutes ses réservations associées.

Exemple de requête:

```
curl -X DELETE http://localhost:8080/api/events/4
```

Exemple de réponse (HTTP 204): *Aucun contenu retourné, juste un code de statut 204 en cas de succès.*

En cas d'événement non trouvé (HTTP 404):

```
{
  "error": "Événement non trouvé"
}
```

API des Réservations

1. Lister toutes les réservations

Endpoint: `GET /api/bookings`

Description: Récupère la liste de toutes les réservations effectuées.

Exemple de requête:

```
curl -X GET http://localhost:8080/api/bookings
```

Exemple de réponse:


```
[
  {
    "id": 1,
    "event_id": 1,
    "customer_name": "Jean Dupont",
    "seats_booked": 3,
    "booking_date": "2023-05-15T14:30:00.000Z",
    "event_name": "Concert de Jazz"
  },
  {
    "id": 2,
    "event_id": 2,
    "customer_name": "Marie Martin",
    "seats_booked": 2,
    "booking_date": "2023-05-15T15:10:00.000Z",
    "event_name": "Conférence Tech"
  }
]
```

2. Récupérer une réservation spécifique

Endpoint: `GET /api/bookings/{id}`

Description: Récupère les détails d'une réservation spécifique par son ID.

Exemple de requête:

```
curl -X GET http://localhost:8080/api/bookings/1
```

Exemple de réponse:

```
{
  "id": 1,
  "event_id": 1,
  "customer_name": "Jean Dupont",
  "seats_booked": 3,
  "booking_date": "2023-05-15T14:30:00.000Z",
  "event_name": "Concert de Jazz"
}
```

En cas de réservation non trouvée (HTTP 404):

```
{
  "error": "Réservation non trouvée"
}
```

3. Voir les réservations pour un événement spécifique

Endpoint: `GET /api/events/{eventId}/bookings`

Description: Récupère toutes les réservations pour un événement spécifique.

Exemple de requête:

```
curl -X GET http://localhost:8080/api/events/1/bookings
```

Exemple de réponse:

```
[
  {
    "id": 1,
    "event_id": 1,
    "customer_name": "Jean Dupont",
    "seats_booked": 3,
    "booking_date": "2023-05-15T14:30:00.000Z"
  },
  {
    "id": 3,
    "event_id": 1,
    "customer_name": "Sophie Lefebvre",
    "seats_booked": 5,
    "booking_date": "2023-05-15T16:45:00.000Z"
  }
]
```

4. Créer une nouvelle réservation

Endpoint: `POST /api/bookings`

Description: Crée une nouvelle réservation pour un événement spécifique.

Corps de la requête:

```
{
  "event_id": 1,
  "customer_name": "Jean Dupont",
  "seats_booked": 3
```

```
}
```

Exemple de requête:

```
curl -X POST http://localhost:8080/api/bookings \
  -H "Content-Type: application/json" \
  -d '{
    "event_id": 1,
    "customer_name": "Jean Dupont",
    "seats_booked": 3
  }'
```

Exemple de réponse (HTTP 201):

```
{
  "id": 4,
  "event_id": 1,
  "customer_name": "Jean Dupont",
  "seats_booked": 3,
  "event_name": "Concert de Jazz"
}
```

En cas d'erreur de validation (HTTP 400):

```
{
  "error": "Tous les champs sont requis (event_id, customer_name,
seats_booked) et seats_booked doit être positif"
}
```

En cas de places insuffisantes (HTTP 400):

```
{
  "error": "Pas assez de places disponibles",
  "available": 2,
  "requested": 3
}
```

5. Annuler une réservation

Endpoint: `DELETE /api/bookings/{id}`

Description: Annule une réservation spécifique et libère les places associées.

Exemple de requête:

```
curl -X DELETE http://localhost:8080/api/bookings/1
```

Exemple de réponse (HTTP 204): *Aucun contenu retourné, juste un code de statut 204 en cas de succès.*

En cas de réservation non trouvée (HTTP 404):

```
{
  "error": "Réservation non trouvée"
}
```

Test de Surbooking

La fonctionnalité de gestion du surbooking est un aspect central de l'application. Elle garantit que deux utilisateurs ne peuvent pas réserver les mêmes places simultanément, grâce à l'utilisation de transactions avec verrouillage (`SELECT FOR UPDATE`).

Scénario de test

1. **Événement avec disponibilité limitée**:
    a. L'événement 1 "Concert de Jazz" dispose de 100 places disponibles.
2. **Première réservation importante**:

```
curl -X POST http://localhost:8080/api/bookings \
  -H "Content-Type: application/json" \
  -d '{
    "event_id": 1,
    "customer_name": "Organisation ABC",
    "seats_booked": 80
  }'
```

Résultat attendu:

```
{
  "id": 5,
  "event_id": 1,
  "customer_name": "Organisation ABC",
  "seats_booked": 80,
  "event_name": "Concert de Jazz"
}
```

3. **Tentative de surbooking**:

```
curl -X POST http://localhost:8080/api/bookings \
  -H "Content-Type: application/json" \
  -d '{
    "event_id": 1,
    "customer_name": "Pierre Martin",
    "seats_booked": 30
  }'
```

Résultat attendu (HTTP 400):

```
{
  "error": "Pas assez de places disponibles",
  "available": 20,
  "requested": 30
}
```

4. **Vérification de l'état de l'événement**:

```
curl -X GET http://localhost:8080/api/events/1
```

Résultat attendu:

```
{
  "id": 1,
  "name": "Concert de Jazz",
  "total_seats": 100,
  "available_seats": 20,
  "event_date": "2025-06-15T20:00:00.000Z",
  "created_at": "2023-05-15T10:30:00.000Z",
  "updated_at": "2023-05-15T10:30:00.000Z"
}
```

5. **Réservation des places restantes**:

```
curl -X POST http://localhost:8080/api/bookings \
  -H "Content-Type: application/json" \
  -d '{
    "event_id": 1,
    "customer_name": "Pierre Martin",
    "seats_booked": 20
  }'
```

Résultat attendu:

```
{
  "id": 6,
  "event_id": 1,
  "customer_name": "Pierre Martin",
  "seats_booked": 20,
  "event_name": "Concert de Jazz"
}
```

6. **Vérification de l'événement complet**:

```
curl -X GET http://localhost:8080/api/events/1
```

Résultat attendu:

```
{
  "id": 1,
  "name": "Concert de Jazz",
  "total_seats": 100,
  "available_seats": 0,
  "event_date": "2025-06-15T20:00:00.000Z",
  "created_at": "2023-05-15T10:30:00.000Z",
  "updated_at": "2023-05-15T10:30:00.000Z"
}
```

7. **Tentative de réservation sur un événement complet**:

```
curl -X POST http://localhost:8080/api/bookings \
  -H "Content-Type: application/json" \
  -d '{
    "event_id": 1,
    "customer_name": "Julie Rousseau",
    "seats_booked": 1
```

```
  }'
```

Résultat attendu (HTTP 400):

```
{
  "error": "Pas assez de places disponibles",
  "available": 0,
  "requested": 1
}
```

Conclusion

Cette application backend de réservation d'événements implémente toutes les fonctionnalités essentielles pour gérer les événements et les réservations. La gestion du surbooking via des transactions verrouillées garantit l'intégrité des données même en cas d'accès concurrents.

Pour une application complète, un front-end pourrait être développé en utilisant un framework moderne comme React, Vue.js ou Angular, qui communiquerait avec ces API REST.

L'architecture microservices offre une base solide pour l'évolution future de l'application, permettant de développer et déployer chaque service indépendamment.