HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

KHOA ĐÀO TẠO SAU ĐẠI HỌC

BÁO CÁO BÀI TẬP LỚN

BỘ MÔN: CÁC HỆ THỐNG PHÂN TÁN

Đề Tài: Hệ thống Đặt vé Xem phim Trực tuyến .

Giảng viên	**:**	**TS. Kim Ngọc Bách**
Lớp	**:**	**M25CQHT01-B**
Nhóm thực hiện	**:**	**Nhóm 3**
Danh sách thành viên	**:**	**Hoàng Đức Hùng B25CHHT026**
		Nguyễn Hồng Sơn B25CHHT049
		Vũ Thanh Thiên B25CHHT055

I. TỔNG QUAN CHUNG

1.1. Mô tả Hệ thống

Hệ thống Đặt vé Xem phim Trực tuyến là nền tảng cho phép người dùng đặt vé xem phim online qua các bước cơ bản: đăng nhập, chọn suất chiếu, chọn ghế, thanh toán và nhận vé điện tử. Hệ thống này được xây dựng để phục vụ số lượng người dùng lớn đồng thời, đặc biệt trong các khung giờ cao điểm khi các bộ phim "bom tấn" được công chiếu. Mỗi người dùng có thể dễ dàng truy cập hệ thống từ bất kỳ thiết bị nào, thực hiện các bước đặt vé một cách nhanh chóng và an toàn.

Hệ thống được thiết kế dưới dạng **Microservices Architecture**, tách biệt thành các dịch vụ độc lập, mỗi dịch vụ đảm nhận một chức năng riêng biệt. Mô hình kiến trúc này bao gồm các dịch vụ chuyên biệt như xác thực người dùng, quản lý thông tin phim và ghế, xử lý đặt vé, thanh toán, và gửi thông báo. Điều này giúp hệ thống linh hoạt, dễ mở rộng và chịu lỗi tốt. Khi một dịch vụ có vấn đề, các dịch vụ khác vẫn có thể tiếp tục hoạt động bình thường, đảm bảo tính liên tục của hệ thống. Ngoài ra, kiến trúc này cho phép các nhóm phát triển làm việc độc lập trên các dịch vụ khác nhau, tăng tốc độ phát triển và triển khai các tính năng mới.

1.2. Bối Cảnh và Nhu Cầu

Trong bối cảnh hiện nay, các hệ thống đặt vé trực tuyến phải đối mặt với những thách thức lớn về kỹ thuật và quản lý. Số lượng người dùng truy cập đồng thời liên tục tăng cao, đặc biệt khi có những bộ phim lớn được công chiếu hoặc vào các dịp lễ tết. Những thách thức này không chỉ ảnh hưởng đến trải nghiệm người dùng mà còn đặt ra những yêu cầu khắt khe về hiệu năng, tin cậy và bảo mật của hệ thống.

Các hệ thống đặt vé trực tuyến phải đối mặt với những thách thức cụ thể sau:

- **Số lượng người dùng lớn**: Lúc cao điểm, hàng nghìn người cùng lúc truy cập hệ thống và thực hiện các giao dịch một cách đồng thời. Điều này yêu cầu hệ thống phải có khả năng xử lý lưu lượng truy cập khổng lồ mà không làm giảm tốc độ phản hồi hoặc gây ra sự gián đoạn dịch vụ.
- **Tranh chấp tài nguyên (Race Condition)**: Nhiều người có thể cùng chọn một ghế tại cùng thời điểm, và hệ thống phải đảm bảo rằng mỗi ghế chỉ được bán cho một người dùng duy nhất. Nếu không xử lý đúng cách, có thể dẫn đến tình trạng bán một ghế cho nhiều người, gây ra tranh cãi và mất uy tín của dịch vụ.
- **Tính tin cậy và đảm bảo dữ liệu**: Lỗi trong quá trình thanh toán không được phép làm mất dữ liệu hoặc tạo ra tình trạng không nhất quán. Ví dụ, nếu hệ thống đã trừ tiền từ tài khoản người dùng nhưng lỗi xảy ra trước khi lưu vé, thì hệ thống phải có cơ chế để hoàn tiền hoặc hoàn tất giao dịch. Điều này yêu cầu hệ thống phải xử lý giao dịch phân tán một cách an toàn và chính xác.
- **Sự độc lập của các thành phần**: Lỗi của một dịch vụ không nên ảnh hưởng toàn bộ hệ thống. Ví dụ, nếu dịch vụ gửi email bị lỗi, người dùng vẫn nên nhận được thông báo xác nhận đặt vé thành công, và email có thể được gửi lại sau. Hệ thống phải được thiết kế sao cho nó có khả năng chịu lỗi từng phần mà không làm ảnh hưởng đến tính liên tục hoạt động.
- **Khả năng mở rộng**: Khi số lượng người dùng tăng cao hoặc nhu cầu đối với các dịch vụ thay đổi, hệ thống phải có thể dễ dàng mở rộng mà không cần phải thiết kế lại toàn bộ kiến trúc. Các

dịch vụ riêng lẻ có thể được mở rộng một cách độc lập dựa trên nhu cầu cụ thể, tối ưu hóa việc sử dụng tài nguyên hệ thống.

Những thách thức trên chính là các bài toán điển hình của **Hệ thống Phân tán** - một lĩnh vực quan trọng trong khoa học máy tính hiện đại. Hệ thống phân tán liên quan đến việc phối hợp các máy tính hoặc dịch vụ độc lập để cùng nhau đạt được một mục tiêu chung, trong khi phải xử lý các vấn đề như trì hoãn mạng, lỗi từng phần, và sự không đồng bộ. Việc áp dụng các nguyên lý và kỹ thuật của hệ thống phân tán vào thiết kế và phát triển hệ thống đặt vé trực tuyến sẽ giúp giải quyết những thách thức này một cách hiệu quả và bền vững.

1.3. Mục Tiêu Đề Tài

Đề tài này được xây dựng với mục tiêu kép vừa là học thuật vừa là kỹ thuật, nhằm kết hợp lý thuyết với thực hành để tạo ra một sản phẩm có giá trị cao.

Mục tiêu học thuật:

- **Áp dụng các khái niệm lý thuyết về Hệ thống Phân tán vào bài toán thực tế**: Đề tài này sẽ giúp sinh viên hiểu rõ cách các khái niệm lý thuyết như loại trừ tương hỗ (Mutual Exclusion), giao dịch phân tán (Distributed Transaction), và khả năng chịu lỗi (Fault Tolerance) được áp dụng vào một bài toán thực tế. Thay vì chỉ học lý thuyết trên lớp, sinh viên sẽ thấy rõ được cách các khái niệm này được triển khai trong một hệ thống sản xuất thực tế.
- **Hiểu rõ mối liên hệ giữa lý thuyết và công nghệ triển khai**: Đề tài sẽ giúp sinh viên nhận thấy rằng các công nghệ như Redis, RabbitMQ, và Saga Pattern không phải chỉ là những công cụ ngẫu nhiên, mà chúng là những giải pháp cụ thể để giải quyết các vấn đề lý thuyết trong hệ thống phân tán. Hiểu được mối liên hệ này sẽ giúp sinh viên đưa ra những quyết định thiết kế tốt hơn trong tương lai.

Mục tiêu kỹ thuật:

- **Thiết kế hệ thống theo kiến trúc Microservices**: Sinh viên sẽ học cách chia hệ thống thành các dịch vụ độc lập, thiết kế các giao diện giữa chúng, và quản lý sự phối hợp giữa các dịch vụ. Kỹ năng này rất quan trọng trong phát triển phần mềm hiện đại, vì ngày càng có nhiều công ty lớn chuyển sang kiến trúc Microservices.
- **Giải quyết vấn đề tranh chấp tài nguyên (Race Condition)**: Sinh viên sẽ hiểu được cách sử dụng các cơ chế khóa phân tán (Distributed Lock) như Redis để đảm bảo rằng chỉ một người dùng có thể đặt một ghế cụ thể. Kỹ năng này có thể được áp dụng cho nhiều bài toán khác ngoài đặt vé, chẳng hạn như quản lý hàng tồn kho hoặc các tài nguyên hữu hạn khác.
- **Xử lý giao dịch phân tán (Distributed Transaction)**: Sinh viên sẽ tìm hiểu cách sử dụng Saga Pattern để đảm bảo tính nhất quán của dữ liệu khi một giao dịch trải dài trên nhiều dịch vụ khác nhau. Điều này bao gồm cơ chế hoàn lại (Rollback) khi có lỗi xảy ra giữa các bước của giao dịch.
- **Đảm bảo khả năng chịu lỗi và mở rộng hệ thống**: Sinh viên sẽ học cách xây dựng hệ thống sao cho nó có thể tiếp tục hoạt động ngay cả khi một số thành phần gặp sự cố, và cách để hệ thống có thể được mở rộng một cách dễ dàng khi nhu cầu tăng lên.

1.4. Phạm vi Đề Tài

Để quản lý độ phức tạp và đảm bảo tính khả thi trong phạm vi của một bài tập lớn cuối kỳ, đề tài này được giới hạn trong những phạm vi cụ thể. Hệ thống sẽ được xây dựng với năm microservices, nhưng một số trong số chúng sẽ được đơn giản hóa bằng cách dữ liệu hoặc chức năng để tập trung vào các khái niệm chính.

Phạm vi bao gồm:

- Xây dựng năm microservices chính: Auth Service (xác thực), Cinema Service (quản lý phim và ghế), Booking Service (đặt vé), Payment Service (thanh toán), và Notification Service (gửi thông báo).
- Giải quyết vấn đề tranh chấp tài nguyên thông qua Redis Distributed Lock để đảm bảo mỗi ghế chỉ được bán cho một người dùng.
- Xử lý giao dịch phân tán bằng Saga Orchestration Pattern để đảm bảo tính nhất quán khi có lỗi trong quá trình thanh toán.
- Xây dựng truyền thông bất đồng bộ thông qua RabbitMQ để gửi vé điện tử mà không làm chậm luồng đặt vé chính.
- Triển khai hệ thống bằng Docker để dễ dàng chạy và kiểm thử các dịch vụ.

Phạm vi không bao gồm (được hoặc đơn giản hóa):

- Xác thực người dùng thực tế: Hệ thống sẽ sử dụng JWT token thay vì tích hợp với hệ thống xác thực thực.
- Dữ liệu phim và ghế thực: Dữ liệu sẽ được hardcoded hoặc được để tránh phức tạp của việc quản lý cơ sở dữ liệu phim thực.
- Thanh toán thực: Payment Service sẽ giả lập quá trình thanh toán thay vì tích hợp với các cổng thanh toán thực tế như Stripe hoặc PayPal.
- Email thực: Notification Service có thể sử dụng Gmail test hoặc mail server thay vì email production.
- Các tính năng nâng cao như Circuit Breaker, Service Mesh, hoặc API Gateway sẽ không được triển khai trong phần này, nhưng sẽ được đề cập đến trong phần hướng phát triển.

II. KIẾN TRÚC TỔNG THỂ HỆ THỐNG

Hệ thống được thiết kế theo kiến trúc **Microservices (SOA)**, bao gồm **5 dịch vụ độc lập**, mỗi dịch vụ chuyên biệt xử lý một chức năng cụ thể. Các dịch vụ này có thể được phát triển, triển khai và mở rộng một cách độc lập, giúp hệ thống linh hoạt và chịu lỗi tốt.

2.1. Danh sách các Microservices

STT	Service	Vai trò	Tính chất
1	Auth Service	Xác thực người dùng ()	
2	Cinema Service	Cung cấp thông tin phim, ghế ()	
3	Booking Service	Xử lý đặt vé, giữ ghế, thanh toán	Full
4	Payment Service	Giả lập thanh toán	
5	Notification Service	Gửi email và hiển thị vé	Full

Mô tả chi tiết:

Auth Service - Xác thực người dùng và cấp phát JWT Token. Đây là service , chỉ dùng để mô phỏng quy trình đăng nhập mà không cần database thực. Các service khác sẽ xác minh token này để đảm bảo người dùng đã được xác thực.

Cinema Service - Cung cấp thông tin phim, danh sách suất chiếu, sơ đồ ghế, và giá vé. Đây cũng là service với dữ liệu được hardcoded, tập trung chủ yếu vào việc mô phỏng dữ liệu phim thay vì quản lý database thực.

Booking Service - Service trung tâm xử lý toàn bộ quy trình đặt vé, từ khóa ghế → lưu booking → gọi thanh toán → cập nhật trạng thái. Đây là service có logic phức tạp nhất, chứa tất cả quy tắc kinh doanh liên quan đến đặt vé.

Payment Service - Giả lập quá trình thanh toán, trả về kết quả SUCCESS hoặc FAIL để test các trường hợp thất bại trong giao dịch. Đây là service , không thực sự xử lý tiền thật.

Notification Service - Xử lý gửi email vé điện tử cho khách hàng. Đây là service full, thực tế gửi email qua SMTP dựa trên template HTML.

2.2. Các Công Nghệ Giao Tiếp

Các service giao tiếp với nhau thông qua ba kênh chính:

Công Nghệ	Loại Giao Tiếp	Mục đích sử dụng	Ví dụ trong hệ thống
REST API	Đồng bộ (Synchronous)	Cần kết quả tức thời	Booking Service gọi Payment Service, Cinema Service
RabbitMQ	Bất đồng bộ (Asynchronous)	Tác vụ không cần phối hợp chặt chẽ	Booking Service gửi event đến Notification Service
Redis	Distributed Lock	Khóa tài nguyên chia sẻ	Khóa ghế khi user đặt vé

Chi tiết từng công nghệ:

REST API (Đồng bộ)

- Được sử dụng khi một service cần nhận kết quả từ service khác ngay lập tức để tiếp tục xử lý
- Ví dụ: Khi Booking Service cần lấy thông tin ghế từ Cinema Service hoặc gọi thanh toán từ Payment Service
- Đặc điểm: Service phải chờ phản hồi, nếu service khác bị chậm thì cả luồng sẽ bị chậm

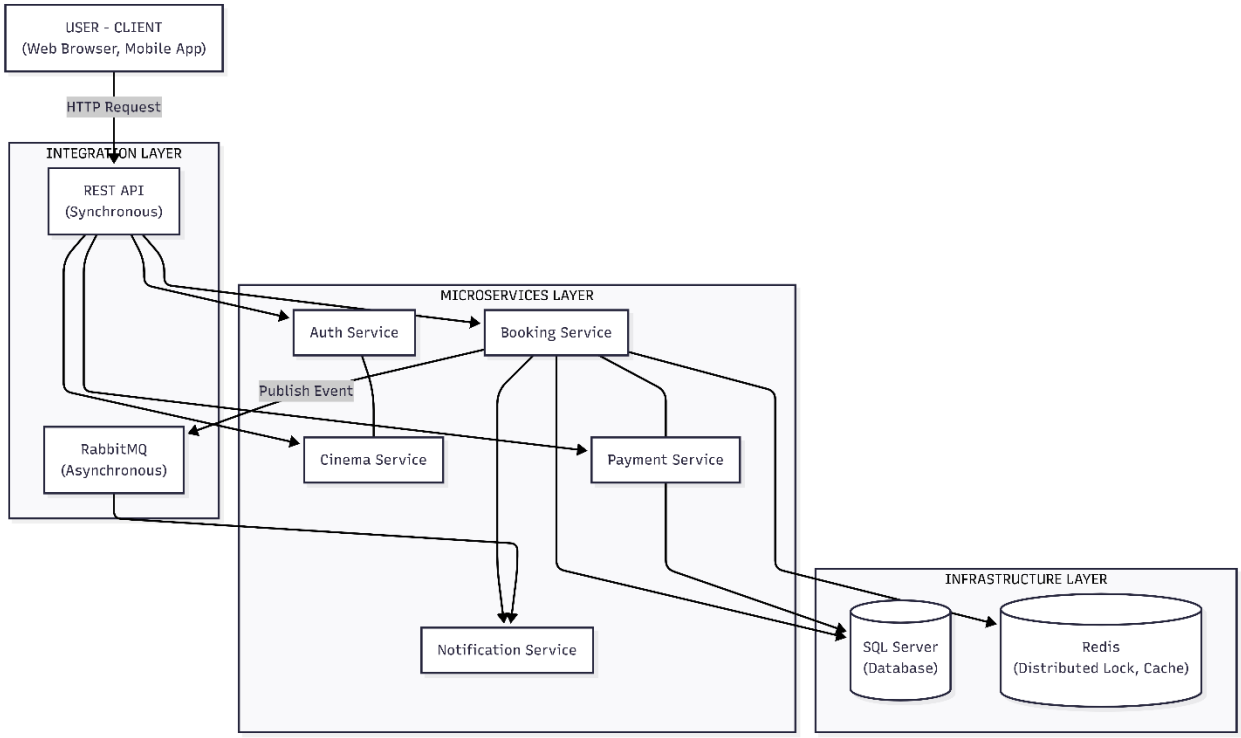
RabbitMQ (Bất đồng bộ)

- Được sử dụng khi một service muốn thông báo cho service khác về một sự kiện, nhưng không cần chờ phản hồi
- Ví dụ: Sau khi thanh toán thành công, Booking Service gửi event qua RabbitMQ để Notification Service gửi email
- Đặc điểm: Booking Service hoàn tất giao dịch ngay, Notification Service xử lý email sau. Nếu Notification Service bị lỗi, message vẫn được giữ lại trong queue

Redis (Distributed Lock)

- Được sử dụng để khóa ghế tạm thời khi user đặt vé, tránh tình trạng bán trùng ghế
- Cơ chế: Sử dụng lệnh SETNX để đặt lock, nếu thành công thì user có quyền đặt ghế này
- Thời hạn: Lock tự động hết hạn sau 600 giây (10 phút), nếu user không thanh toán thì ghế được giải phóng

2.3. Sơ đồ Kiến Trúc Hệ Thống



III. PHÂN TÍCH CHI TIẾT YÊU CẦU

3.1. Auth Service (Xác thực)

Mục đích:

- Giả lập quá trình đăng nhập người dùng
- Cung cấp thông tin User ID cho các service khác

Luồng hoạt động:

Bước	Chi tiết
1	Người dùng gửi yêu cầu đăng nhập (email, password)
2	Service kiểm tra thông tin ()
3	Trả về JWT token giả lập và thông tin người dùng

Yêu cầu chi tiết:

Thông số	Chi tiết
Endpoint	POST /auth/login
Input	{ email, password }
Output	{ userId, token, role }
Database	Không cần - chỉ dữ liệu
Response Status	200 (Success), 401 (Invalid credentials)

Ý nghĩa phân tán:

- Mô phỏng **Remote Authentication** (Xác thực từ xa)
- Tách biệt chức năng xác thực khỏi logic đặt vé
- Các service phụ thuộc vào Auth Service

3.2. Cinema Service (Thông tin Phim & Rạp)

Mục đích:

- Cung cấp danh sách phim, suất chiếu, và sơ đồ ghế
- Trả về trạng thái ghế (trống, đã đặt)

Luồng hoạt động:

Bước	Chi tiết
1	Người dùng gọi API lấy danh sách phim
2	Người dùng chọn suất chiếu
3	Service trả về danh sách ghế (hardcoded)
4	Người dùng chọn ghế để đặt

Yêu cầu chi tiết - Endpoints:

Endpoint 1: Lấy danh sách phim

```
GET /cinema/movies
Output: [
  { movieId: 1, title: "Avatar 2", duration: 192 },
  { movieId: 2, title: "The Marvels", duration: 110 }
]
```

Endpoint 2: Lấy danh sách suất chiếu

```
GET /cinema/movies/{movieId}/showtimes
Output: [
  { showId: 1, time: "14:00", movieId: 1, totalSeats: 100 },
  { showId: 2, time: "18:00", movieId: 1, totalSeats: 100 }
]
```

Endpoint 3: Lấy sơ đồ ghế

```
GET /cinema/shows/{showId}/seats
Output: [
  { seatCode: "A1", status: "AVAILABLE", price: 100000 },
  { seatCode: "A2", status: "BOOKED", price: 100000 },
  ...
]
```

Tính năng	Chi tiết
Dữ liệu	Lấy từ database
Trạng thái ghế	AVAILABLE, BOOKED, PAID , HOLD
Giá vé	Cố định per endpoint
Caching	Có thể cache trong Redis

Ý nghĩa phân tán:

- **Loose Coupling**: Booking Service không cần biết chi tiết cách Cinema lưu ghế
- Dữ liệu phim/ghế được tập trung tại một service → dễ quản lý và cập nhật
- Cinema Service là **Source of Truth** cho thông tin ghế
- Nếu Cinema Service tạm thời bị chậm, không ảnh hưởng đến booking đã được lưu

3.3. Booking Service (Xử lý Đặt Vé - SERVICE TRUNG TÂM)

Mục đích:

- Xử lý quy trình đặt vé từ A đến Z
- Điều phối các service khác (Payment, Notification)
- Đảm bảo tính nhất quán dữ liệu

Luồng nghiệp vụ chi tiết (Quan trọng nhất):

Bước 1: Người dùng gửi yêu cầu đặt vé

```
POST /booking/book
Headers: { Authorization: "Bearer <token>" }
Body: {
  userId: 1,
  showId: 1,
  seatCode: "A1"
}
```

Yêu cầu	Chi tiết
Authentication	JWT Token bắt buộc
Validation	userId, showId, seatCode không được rỗng
Response	Trả về bookingId nếu thành công

Bước 2: Khóa Ghế bằng Redis (Distributed Lock)

Mục đích: Tránh tranh chấp (Race Condition) - đảm bảo chỉ một người được đặt một ghế

Logic:

```
SETNX lock:seat:A1 userId EX 600
```

Kết quả	Xử lý
Thành công	Ghế được khóa, tiếp tục quy trình
Thất bại	Ghế đã bị khóa bởi người khác → Trả lỗi "Seat is locked"

Thời hạn khóa: 600 giây (10 phút)

- Nếu quá 10 phút mà chưa thanh toán → Tự động giải phóng ghế
- User có thể thử đặt ghế khác hoặc cố gắng lại

Ý nghĩa lý thuyết:

- Áp dụng **Loại trừ tương hỗ (Mutual Exclusion)** - chỉ một tiến trình truy cập tài nguyên tại một thời điểm
- Sử dụng **Pessimistic Locking** (Khóa bi quan) - khóa trước khi thay đổi dữ liệu

Bước 3: Lưu Booking với trạng thái HOLD

Mục đích: Đảm bảo không mất dữ liệu nếu các bước sau gặp lỗi

Dữ liệu lưu vào SQL Server:

```
INSERT INTO Bookings (
  bookingId, userId, showId, seatCode,
  status, createdAt, paidAt
)
VALUES (
  'BK001', 1, 1, 'A1',
  'PENDING', NOW(), NULL
)
```

Các trạng thái Booking:

Trạng thái	Ý nghĩa
HOLD	Ghế đang được giữ, chưa thanh toán
BOOKED	Thanh toán thành công, vé chính thức
EXPRIED	Trạng thái ghế được giải phóng sau khi bị lock

Bước 4: Gọi Payment Service (Saga Step 1)

Logic:

```
POST /payment/pay
Body: { bookingId: 'BK001', amount: 100000, userId: 1 }
```

Response	Xử lý tiếp theo
SUCCESS	Tiếp tục bước 5
FAIL	Thực hiện Compensating Transaction (hoàn lại)

Ý nghĩa lý thuyết:

- Áp dụng **Giao dịch Phân tán (Distributed Transaction)**
- Sử dụng **Saga Pattern** (Orchestration) - điều phối nhiều service để hoàn thành giao dịch

Bước 5: Cập nhật trạng thái Booking → PAID

```
UPDATE Bookings
SET status = 'PAID', paidAt = NOW()
WHERE bookingId = 'BK001'
```

Các hành động đi kèm:

- Xóa lock ghế khỏi Redis: DEL lock:seat:A1
- Cập nhật Cinema Service để ghế chuyển sang "BOOKED"
- Chuẩn bị dữ liệu gửi thông báo

Bước 6: Gửi sự kiện qua RabbitMQ

Mục đích: Gửi vé điện tử mà không làm chậm luồng đặt vé

Message được gửi:

```
{
 "bookingId": "BK001",
 "userId": 1,
 "email": "user@gmail.com",
 "movieTitle": "Avatar 2",
 "showTime": "18:00",
 "seatCode": "A1"
}
```

Ý nghĩa lý thuyết:

- Áp dụng **Truyền thông Bất đồng bộ (Asynchronous Communication)**
- Tách biệt **Luồng chính** (Booking Service hoàn tất ngay) và **Tác vụ phụ** (Gửi email sau)

Xử lý lỗi thanh toán (Compensating Transaction):

Nếu bước 4 (Payment) thất bại:

Bước Rollback	Chi tiết
1	Cập nhật status = 'CANCELLED'
2	Xóa lock ghế khỏi Redis: DEL lock:seat:A1
3	Gửi thông báo lỗi cho người dùng
4	Kết thúc giao dịch

Data Model - SQL Server:

```
CREATE TABLE Bookings (
```

```
    bookingId NVARCHAR(20) PRIMARY KEY,
    userId INT NOT NULL,
    showId INT NOT NULL,
    seatCode NVARCHAR(5) NOT NULL,
    status NVARCHAR(20) NOT NULL,
    createdAt DATETIME NOT NULL,
    paidAt DATETIME,
    paidAmount INT
);
```

```
CREATE TABLE Seats (
    showId INT,
    seatCode NVARCHAR(5),
    status NVARCHAR(20), -- AVAILABLE, BOOKED, RESERVED
    PRIMARY KEY (showId, seatCode)
);
```

3.4. Payment Service (Thanh Toán)

Mục đích:

- Giả lập xử lý thanh toán để demo Saga Pattern
- Không cần lưu DB, chỉ logic
- Giúp kiểm tra các trường hợp lỗi trong giao dịch

Luồng hoạt động:

```
POST /payment/pay
Body: { bookingId, amount, userId }
```

Logic:

1. Kiểm tra bookingId hợp lệ

2. Sleep 1 giây (giả lập xử lý)

3. Trả về SUCCESS hoặc FAIL (10% thất bại)

Yêu cầu chi tiết:

Tính năng	Yêu cầu
Thời gian xử lý	~1 giây ()
Tỷ lệ thất bại	~10% (để test Saga)

Tính năng	Yêu cầu
Response	{ status: SUCCESS/FAIL, transactionId, timestamp }
Database	Không cần lưu dữ liệu thanh toán
Validation	Kiểm tra bookingId tồn tại

Ý nghĩa phân tán:

- Mô phỏng một dịch vụ thanh toán thực tế nhưng đơn giản hóa
- Giúp test Compensating Transaction khi thanh toán thất bại
- Tách biệt logic thanh toán khỏi logic đặt vé

3.5. Notification Service (Gửi Vé Điện Tử)

Mục đích:

- Nhận sự kiện từ RabbitMQ
- Gửi email vé điện tử cho khách hàng
- Đảm bảo độ tin cậy (Retry nếu gửi thất bại)

Luồng hoạt động:

Bước 1: Nhận message từ RabbitMQ

- Lắng nghe Queue: booking-notification
- Lấy các thông tin: bookingId, email, movieTitle, seatCode, v.v.
- Xử lý message một cách tuần tự (Sequential processing)

Bước 2: Lấy thông tin vé chi tiết

```
GET /booking/{bookingId}
Response: {
  bookingId, movieTitle, showTime, seatCode,
  customerName, totalPrice
}
```

Dữ liệu cần	Chi tiết
Tên khách hàng	Từ booking hoặc user service
Tên phim	Từ cinema service
Giờ chiếu	Từ cinema service
Mã ghế	Từ booking
Giá vé	Từ cinema service

Bước 3: Render vé HTML

Sử dụng **Thymeleaf** để tạo template vé điện tử:

```
<div class="ticket">
  <h2>Vé Xem Phim Điện Tử</h2>
  <p><strong>Phim:</strong> Avatar 2</p>
  <p><strong>Suất chiếu:</strong> 2024-01-15 18:00</p>
  <p><strong>Ghế:</strong> A1</p>
  <p><strong>Giá vé:</strong> 100.000 VNĐ</p>
  <img src="..." /> <!-- QR Code -->
</div>
```

Bước 4: Gửi Email qua Gmail SMTP

```
FROM: system@cinema.vn
TO: user@gmail.com
SUBJECT: Vé xem phim của bạn
BODY: HTML content từ Thymeleaf
ATTACHMENTS: (Tuỳ chọn)
```

Yêu cầu chi tiết:

Tính năng	Yêu cầu
Message Queue	RabbitMQ (Queue: booking-notification)
Email Server	Gmail SMTP (smtp.gmail.com:587)
Template Engine	Thymeleaf HTML

Tính năng	Yêu cầu
QR Code	Tạo QR code từ bookingId
Retry Logic	Thử lại 3 lần nếu gửi thất bại
Dead Letter Queue	Nếu vẫn thất bại sau 3 lần
Timeout	30 giây (timeout cho email)

Ý nghĩa lý thuyết:

- **Fault Tolerance**: Hệ thống tiếp tục hoạt động ngay cả khi Notification Service thất bại
- **Idempotent**: Gửi lại email cùng bookingId không tạo vé trùng lặp (kiểm tra đã gửi chưa)
- **Asynchronous**: Không làm chậm luồng đặt vé (user nhận booking confirmation ngay)

IV. CÁC LUỒNG GIAO DỊCH CHÍNH

4.1. Luồng Thành Công (Happy Path)

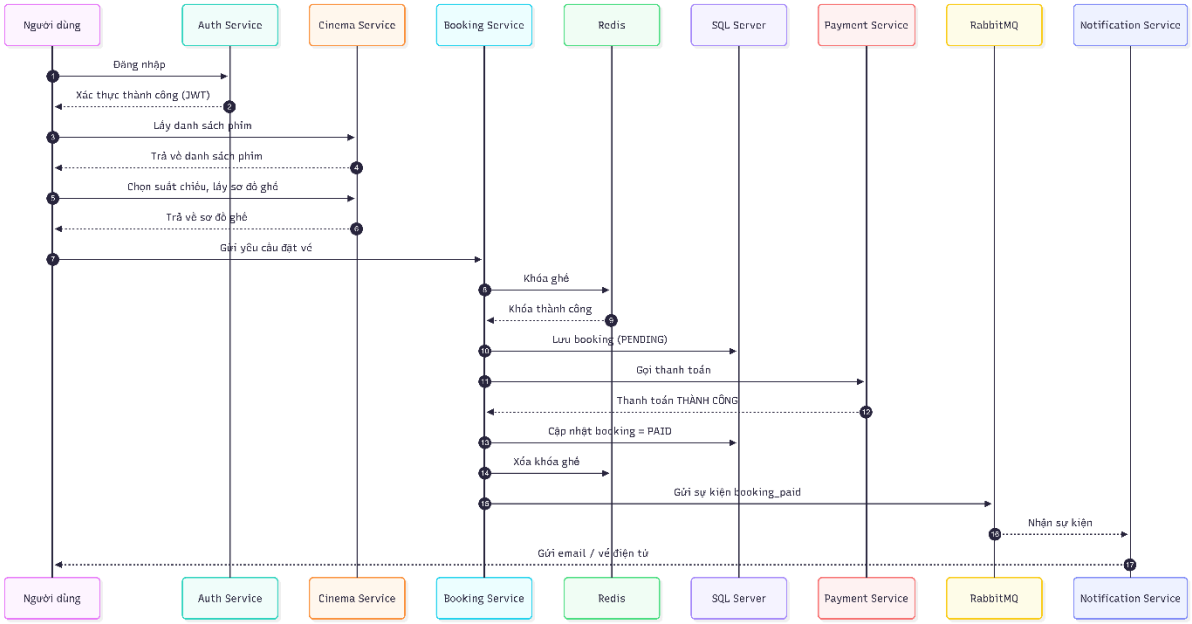
Đây là trường hợp lý tưởng khi tất cả các bước đều thực hiện thành công, từ khi người dùng đặt vé cho đến khi nhận được vé điện tử.

Các bước chi tiết:

Bước	Tác vụ	Kết quả
1	Người dùng đăng nhập	Nhận JWT token từ Auth Service
2	Lấy danh sách phim	Từ Cinema Service
3	Chọn suất chiếu	Lấy sơ đồ ghế từ Cinema Service
4	Gửi yêu cầu đặt vé	POST /booking/book
5	Khóa ghế Redis	SETNX lock:seat:A1 → **SUCCESS**

Bước	Tác vụ	Kết quả
6	Lưu booking PENDING	INSERT vào SQL Server
7	Gọi Payment Service	REST call → **SUCCESS**
8	Cập nhật status PAID	UPDATE booking status
9	Xóa lock Redis	DEL lock:seat:A1
10	Gửi event RabbitMQ	Publish message
11	Notification Service nhận	Từ RabbitMQ queue
12	Render vé HTML	Thymeleaf template
13	Gửi email	Gmail SMTP → **SUCCESS**
14	Người dùng nhận vé	Email + confirmation page

Sơ đồ Luồng:



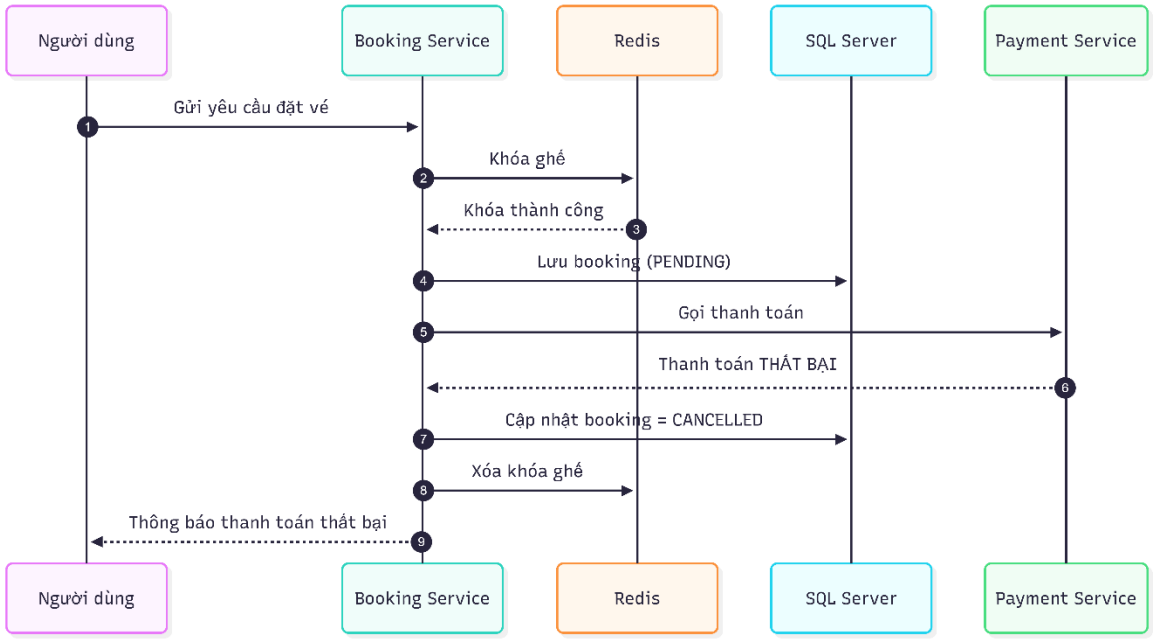
4.2. Luồng Thất Bại (Failure Path - Payment Failed)

Khi thanh toán thất bại, hệ thống phải rollback tất cả các thay đổi để tránh tình trạng không nhất quán.

Kịch bản: Người dùng đặt vé thành công, nhưng Payment Service trả về FAIL

Bước	**Tác vụ**	**Chi tiết**
1-6	Giống luồng thành công	Đến bước lưu PENDING
7	Gọi Payment Service	REST call → **FAIL**
8	Compensating Transaction	Rollback bắt đầu
9	Cập nhật status CANCELLED	UPDATE booking status
10	Xóa lock Redis	DEL lock:seat:A1
11	Gửi thông báo lỗi	REST API response
12	Người dùng thấy lỗi	Display error message

Sơ đồ Luồng:



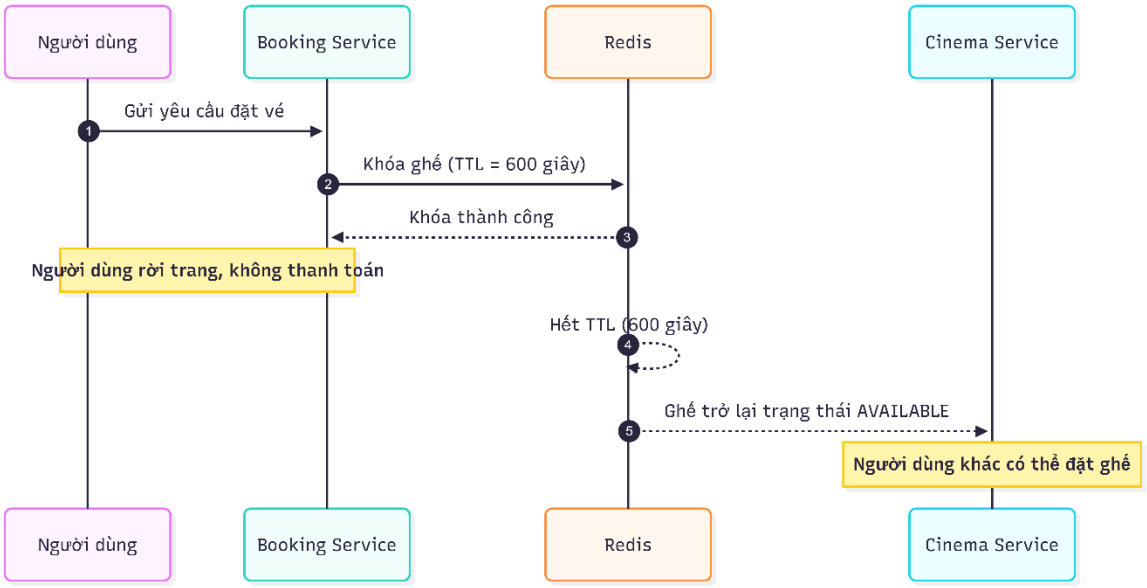
4.3. Luồng Timeout (Hết Thời gian Khóa)

Khi người dùng khóa một ghế nhưng không thanh toán trong 10 phút, khóa tự động hết hạn.

Kịch bản: User A đặt ghế A1, nhưng rời trang mà không thanh toán

Thời gian	**Tác vụ**	**Chi tiết**
T+0s	User A khóa ghế	SETNX lock:seat:A1 EX 600
T+1s	Lưu booking PENDING	Status = PENDING
T+30s	User A rời trang	Không hoàn tất thanh toán
T+600s	Lock hết hạn	Redis tự động xóa lock
T+601s	User B đặt ghế	SETNX lock:seat:A1 → **SUCCESS**

Sơ đồ Timeline:



V. CÁC TRẠNG THÁI NGHIỆP VỤ

Hệ thống đặt vé xem phim sử dụng ba trạng thái chính để quản lý vòng đời của mỗi booking. Mỗi trạng thái đại diện cho một giai đoạn cụ thể trong quá trình giao dịch.

5.1. Danh sách Trạng thái Booking

STT	Trạng thái	Ý nghĩa	Kỳ vọng
1	PENDING	Ghế đang được giữ tạm thời, chờ thanh toán	Chuyển sang PAID hoặc CANCELLED
2	PAID	Thanh toán thành công, vé chính thức	Gửi email vé, không thay đổi
3	CANCELLED	Thanh toán thất bại hoặc người dùng hủy	Giải phóng ghế, không gửi email

5.2. Chi tiết từng Trạng thái

1. Trạng thái PENDING (Đang giữ vé)

Định nghĩa:

- Người dùng đã khóa ghế và lưu booking vào database
- Ghế được Redis lock để tránh người khác đặt
- Chưa thanh toán hoặc đang chờ thanh toán

Thời lượng:

- Tối đa: 600 giây (10 phút)
- Nếu quá 10 phút chưa thanh toán → Lock tự động hết hạn

Dữ liệu lưu:

```
INSERT INTO Bookings (bookingId, userId, showId, seatCode, status, createdAt, paidAt)
VALUES ('BK001', 1, 1, 'A1', 'PENDING', '2024-01-15 18:00:00', NULL)
```

Trường dữ liệu	Giá trị	Ghi chú
bookingId	BK001	ID duy nhất của booking
userId	1	ID người đặt vé
showId	1	ID suất chiếu
seatCode	A1	Mã ghế
status	PENDING	Trạng thái
createdAt	2024-01-15 18:00:00	Thời điểm tạo
paidAt	NULL	Chưa thanh toán
paidAmount	NULL	Chưa có số tiền

Hành động có thể thực hiện:

- ✓ Thanh toán → Chuyển sang PAID
- ✓ Hủy vé → Chuyển sang CANCELLED
- ✓ Timeout (600s) → Tự động CANCELLED

Ý nghĩa phân tán:

- Đảm bảo durability - dữ liệu được lưu trước khi thanh toán
- Nếu service khác bị lỗi, vẫn không mất booking

2. Trạng thái PAID (Đã thanh toán)

Định nghĩa:

- Thanh toán thành công
- Vé chính thức, người dùng có quyền vào xem phim
- Ghế không còn bị khóa (lock đã xóa)

Dữ liệu lưu:

```
UPDATE Bookings
SET status = 'PAID', paidAt = '2024-01-15 18:00:05', paidAmount = 100000
WHERE bookingId = 'BK001'
```

Trường dữ liệu	Giá trị	Ghi chú
bookingId	BK001	Không thay đổi
status	PAID	Thanh toán thành công
paidAt	2024-01-15 18:00:05	Thời điểm thanh toán
paidAmount	100000	Số tiền thanh toán (VNĐ)

Hành động tiếp theo:

- ✓ Gửi email vé điện tử
- ✓ Cập nhật Cinema Service (ghế → BOOKED)
- ✓ Xóa lock Redis

Không thể thực hiện:

- ✗ Hủy vé (đã thanh toán)
- ✗ Chuyển trạng thái khác

Ý nghĩa phân tán:

- Chỉ PAID booking mới được xem là vé hợp lệ

- Dữ liệu PAID immutable (không thay đổi)

3. Trạng thái CANCELLED (Đã hủy)

Định nghĩa:

- Thanh toán thất bại
- Người dùng hủy booking
- Ghế trở lại AVAILABLE cho người khác

Kịch bản 1: Thanh toán thất bại

```
UPDATE Bookings
SET status = 'CANCELLED', cancelReason = 'Payment failed'
WHERE bookingId = 'BK001'
```

Kịch bản 2: Người dùng chủ động hủy

```
UPDATE Bookings
SET status = 'CANCELLED', cancelReason = 'User cancelled'
WHERE bookingId = 'BK001'
```

Kịch bản 3: Timeout (600s)

```
UPDATE Bookings
SET status = 'CANCELLED', cancelReason = 'Lock timeout'
WHERE bookingId = 'BK001'
```

Trường dữ liệu	Giá trị	Ghi chú
status	CANCELLED	Hủy
cancelReason	Payment failed	Lý do hủy
cancelledAt	2024-01-15 18:00:30	Thời điểm hủy

Hành động tiếp theo:

- ✓ Xóa lock Redis: DEL lock:seat:A1
- ✓ Gửi thông báo lỗi cho user
- ✓ Ghế quay lại AVAILABLE

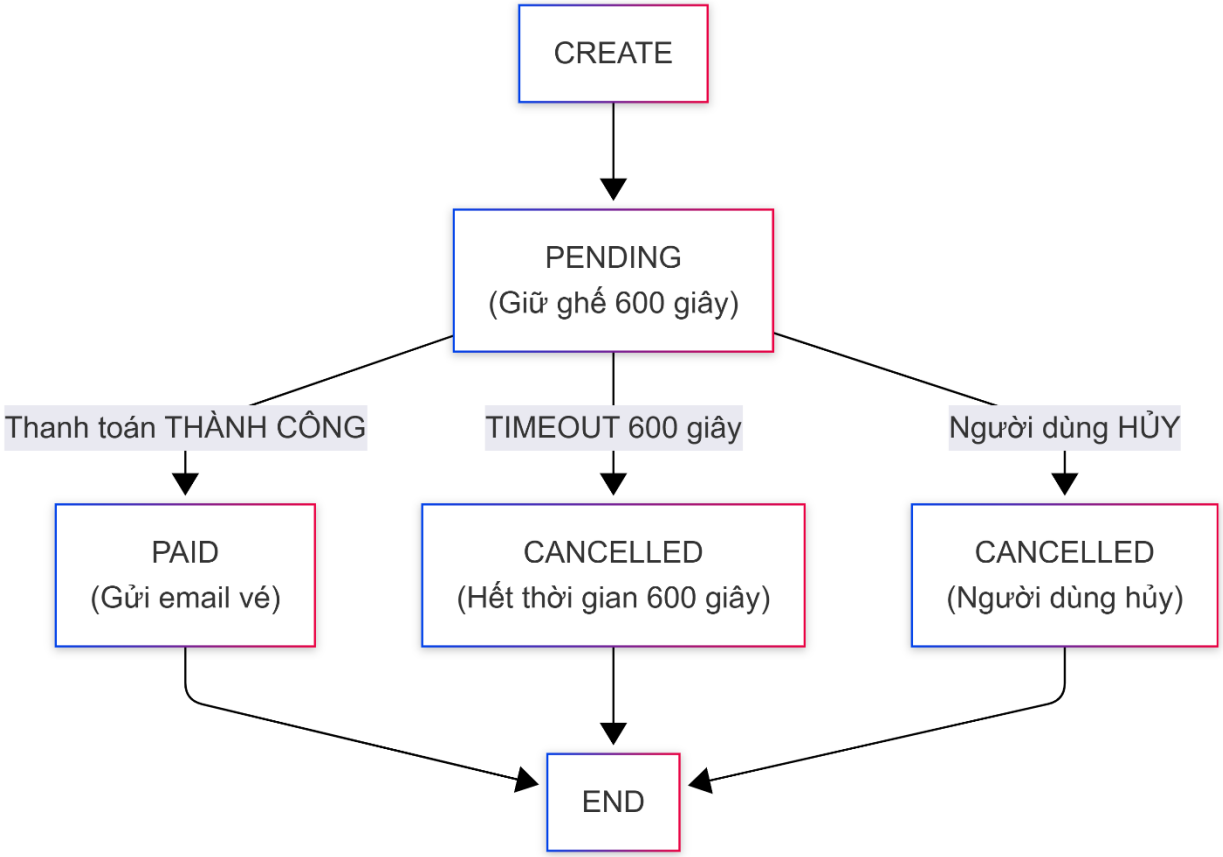
Không thực hiện:

- ✗ Gửi email vé
- ✗ Cập nhật Cinema Service ghế → BOOKED

Ý nghĩa phân tán:

- Compensating Transaction - hoàn tác các thay đổi
- Tài nguyên (ghế) được giải phóng để người khác dùng

5.3. Biểu đồ Chuyển đổi Trạng thái



5.4. Bảng Tóm tắt Trạng thái

Trạng thái	Khi xảy ra	Thời lượng	Lock Redis	Email	Ghế
PENDING	User đặt vé	Max 600s	Active	No	Reserved
PAID	Payment SUCCESS	Vĩnh viễn	Deleted	Yes	Booked

Trạng thái	Khi xảy ra	Thời lượng	Lock Redis	Email	Ghế
CANCELLED	Payment FAIL / Timeout / Cancel	Vĩnh viễn	Deleted	No	Available

---

5.5. Quy tắc Chuyển đổi Trạng thái

Quy tắc 1: PENDING → PAID

- Điều kiện: Payment Service trả SUCCESS
- Hành động: Gửi email, xóa lock
- Không thể hoàn tác

Quy tắc 2: PENDING → CANCELLED (Payment Fail)

- Điều kiện: Payment Service trả FAIL
- Hành động: Gọi Compensating Transaction
- Giải phóng lock và ghế

Quy tắc 3: PENDING → CANCELLED (Timeout)

- Điều kiện: Lock hết hạn sau 600s
- Hành động: Cron job hoặc TTL trigger xóa lock
- Ghế tự động available

Quy tắc 4: PENDING → CANCELLED (User Cancel)

- Điều kiện: User chủ động hủy
- Hành động: Xóa lock, status = CANCELLED
- Ghế available cho người khác

Quy tắc 5: PAID/CANCELLED → No change

- Không thể chuyển từ PAID/CANCELLED sang trạng thái khác
- Đảm bảo consistency

5.6. Impact của từng Trạng thái

PENDING State Impact:

Thành phần	Ảnh hưởng
Redis	Lock active (600s) - chiếm memory
SQL Server	Row được khóa (PENDING)
Cinema Service	Ghế không thể bán cho người khác
User	Chờ thanh toán

Xử lý:

- Redis cleanup: TTL 600s tự động xóa
- SQL cleanup: Cron job xóa PENDING quá 1 tiếng

PAID State Impact:

Thành phần	Ảnh hưởng
SQL Server	Row immutable - không thay đổi
Cinema Service	Ghế = BOOKED - không bán lại
Email	Vé được gửi
User	Có vé chính thức, có thể vào xem

Xử lý:

- SQL data: Archive sau 30 ngày
- Email: Resend nếu user request

CANCELLED State Impact:

Thành phần	Ảnh hưởng
Redis	Lock deleted - giải phóng
SQL Server	Row marked CANCELLED
Cinema Service	Ghế = AVAILABLE - có thể bán lại
User	Không có vé

Xử lý:

- Refund (nếu có): Gửi request tới payment service
- Notification: Gửi email lý do hủy

5.7. Ví dụ Thực tế về Chuyển đổi Trạng thái

Ví dụ 1: Happy Path (PENDING → PAID)

```
Timeline:

18:00:00 - User A lock ghế A1

    INSERT: BK001, status=PENDING

    Redis: SET lock:seat:A1 (EX 600)


18:00:02 - Payment SUCCESS

    UPDATE: BK001, status=PAID, paidAt=18:00:02

    Redis: DEL lock:seat:A1

    Email: Gửi vé


18:00:05 - User A nhận email vé

    Có thể vào xem phim
```

Ví dụ 2: Payment Fail (PENDING → CANCELLED)

```
Timeline:

18:00:00 - User B lock ghế B2

    INSERT: BK002, status=PENDING

    Redis: SET lock:seat:B2
```

18:00:03 - Payment FAIL

Compensating Transaction:

UPDATE: BK002, status=CANCELLED

Redis: DEL lock:seat:B2

Email: Gửi thông báo lỗi

18:00:04 - Ghế B2 available cho User C

Ví dụ 3: Timeout (PENDING → CANCELLED)

Timeline:

18:00:00 - User C lock ghế C3

INSERT: BK003, status=PENDING

Redis: SET lock:seat:C3 (EX 600)

18:05:00 - User C rời trang

Không thanh toán

18:10:00 - Lock hết hạn

Redis: Tự động xóa lock:seat:C3

System: UPDATE BK003, status=CANCELLED (async)

Ghế C3 available

5.8. Data Consistency với Trạng thái

Consistency Rules:

Rule	**Đảm bảo**
PENDING → PAID chỉ 1 lần	Không bán lại vé đã thanh toán
PAID/CANCELLED immutable	Không sửa sau hoàn thành
Lock = PENDING status	Ghế khóa ↔ Booking PENDING
No PAID → other states	Vé đã bán không thể hủy

Transaction Safety:

```
BEGIN TRANSACTION
  -- Bước 1: Lock ghế
  SET lock:seat:A1 = userId (Redis)

  -- Bước 2: Lưu booking
  INSERT INTO Bookings (...)

  -- Bước 3: Thanh toán
  CALL PaymentService()

  -- Bước 4: Cập nhật trạng thái
  IF payment_success THEN
    UPDATE Bookings SET status = 'PAID'
  ELSE
    -- Rollback
    UPDATE Bookings SET status = 'CANCELLED'
    DEL lock:seat:A1
  END IF
COMMIT
```