

**PHENIKAA UNIVERSITY**  
**SCHOOL OF COMPUTING**



**COURSE MODULE: SOFTWARE ARCHITECTURE**  
**PROJECT TITLE: BUILDING A BUS TICKET BOOKING SYSTEM**

Instructor: TS. Vũ Quang Dũng  
Group: Lò Bảo Duy (23010096)  
Class: CSE703110-1-2-25 (N02)

Ha Noi, January 30, 2026

## CONTENTS

<i>IMAGE CATALOG</i> .....	3
<i>PREFACE</i> .....	4
<i>ACKNOWLEDGEMENTS</i> .....	5
1. <i>Architectural Design &amp; Implementation</i> .....	7
1.1 System Context Diagram (C4 Model).....	7
1.2 Service Implementation Details.....	8
1.3 Inter-Service Communication .....	11
2. <i>Testing &amp; Verification</i> .....	13
3. <i>Conclusion &amp; Reflection</i> .....	23
3.1 Lessons Learned.....	24
3.2 Future Improvements.....	24
<i>REFERENCES</i> .....	25

## IMAGE CATALOG

Figure 1. Level 1 - System Context Diagram .....	7
Figure 2. Level 3 - Component Diagram.....	7
Figure 3. Unit Test Booking_1 .....	13
Figure 4. Unit Test Booking_2 .....	13
Figure 5. Unit test Booking_3.....	14
Figure 6. Booking Note Test_1.....	14
Figure 7. Booking Note Test_2.....	15
Figure 8. Booking Note Test_3.....	15
Figure 9. Booking Note Test_4.....	16
Figure 10. Bus Company List .....	16
<i>Figure 11. Footer.....</i>	17
Figure 12. Bus List.....	18
Figure 13. Seat Selection .....	18
Figure 14. QR Code.....	22
Figure 15. Booking Success.....	19
Figure 16. Nginx .....	20
Figure 17. Kafka .....	20
Figure 18. Webhooks .....	21
Figure 19. My Tickets _1 .....	21
Figure 20. My Tickets _2 .....	22
Figure 21. My Tickets _3 .....	22
Figure 22. Docker Containers .....	23

## PREFACE

In the context of the rapid development of information technology, the application of software systems to management and service delivery has become an inevitable trend across many areas of modern society. Particularly in the passenger transportation sector, the increasing demand for fast, convenient, and transparent ticket booking requires software systems not only to provide correct functionality but also to ensure scalability, reliability, high performance, and long-term maintainability.

The Software Architecture course plays an important role in equipping students with systematic thinking at the architectural level, including decomposing systems into independent components, selecting appropriate architectural styles, defining communication mechanisms among services, and optimizing quality attributes such as scalability, availability, and maintainability. Through this course, students are exposed to modern architectural approaches, especially Microservices Architecture, which enables the development of flexible and sustainable large-scale systems in real-world environments.

Based on these practical needs, our team selected the topic “**BUILDING A BUS TICKET BOOKING SYSTEM**” for this course project. The project focuses on

requirement analysis, overall architectural design, service decomposition, inter-service communication mechanisms, and system deployment models. Through this process, the team has the opportunity to apply theoretical knowledge to solve real-world problems while strengthening architectural thinking and system design skills. Although we have made considerable efforts to complete this project, due to limitations in time, experience, and research scope, shortcomings are unavoidable. Therefore, we sincerely look forward to receiving valuable feedback and suggestions from our instructor to further improve the quality of this project.

## **ACKNOWLEDGEMENTS**

We would like to express our sincere gratitude to TS. Vũ Quang Dũng, our lecturer for the Software Architecture course, for his dedicated guidance, valuable feedback, and professional support throughout the completion of this project. His knowledge and practical experience have greatly helped us develop a deeper understanding of architectural design principles and successfully accomplish this work.

We would also like to thank the School of Information Technology – Phenikaa University for providing favorable learning conditions, facilities, and academic resources that enabled us to conduct research and complete this project effectively. Finally, we would like to thank our friends and classmates for their support, discussions, and constructive suggestions during the learning and development process. Their encouragement has been a significant motivation for us to complete our tasks.

Despite our best efforts, the project may still contain limitations and mistakes. We highly appreciate any comments and recommendations from our instructor to help us improve and gain better experience for future studies and projects.



# 1. Architectural Design & Implementation

## 1.1 System Context Diagram (C4 Model)

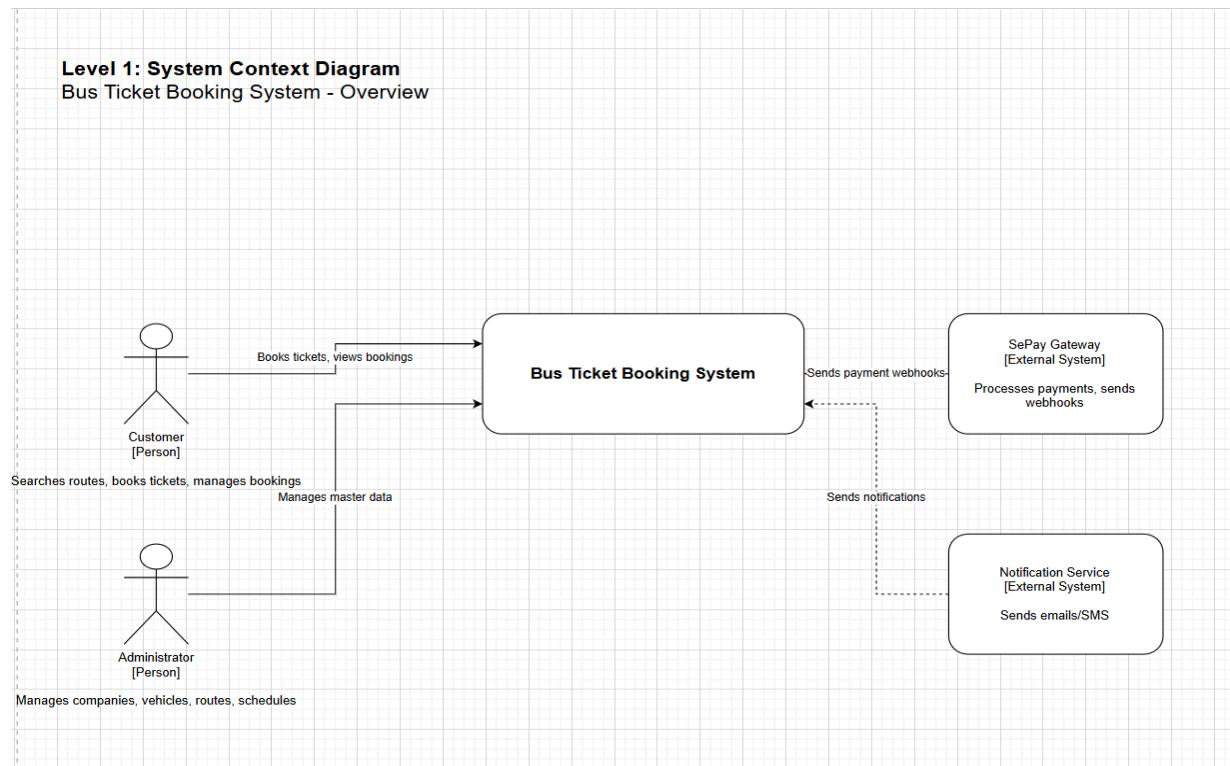


Figure 1. Level 1 - System Context Diagram

**Level 3: Component Diagram**  
Ticket Service - Internal Components

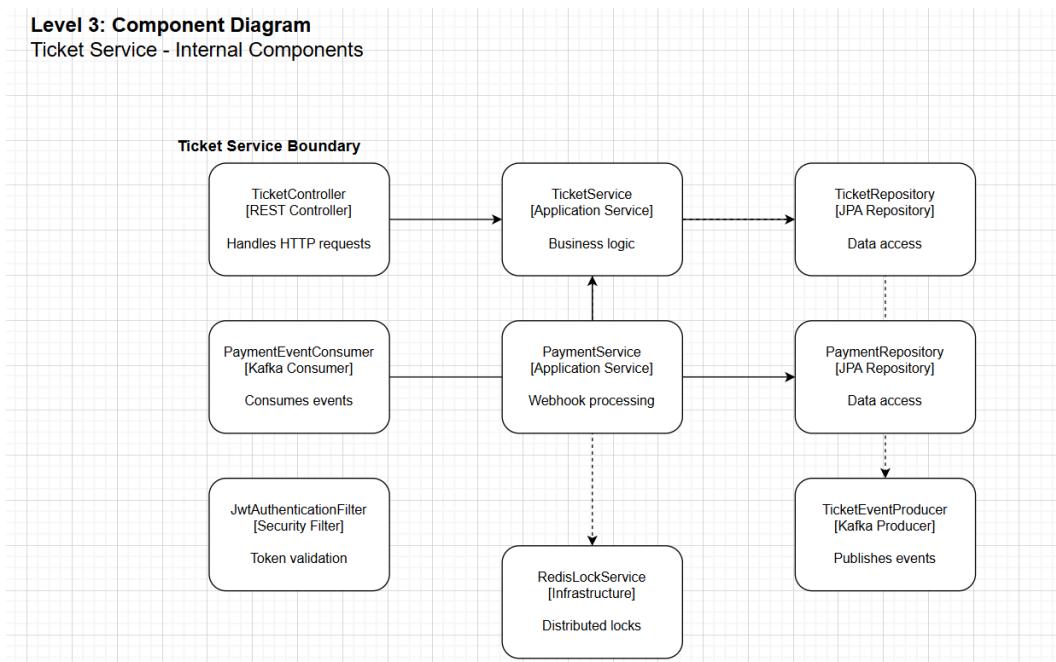


Figure 2. Level 3 - Component Diagram

## 1.2 Service Implementation Details

Each microservice follows a Clean Architecture structure with clear separation of concerns:

```
service/
  └── application/      # Use cases and application services
    └── domain/
      ├── model/        # Domain entities (JPA entities)
      └── repository/   # Repository interfaces
    └── infrastructure/
      ├── config/       # Configuration classes
      ├── exception/    # Exception handlers
      ├── logging/       # Structured logging
      ├── messaging/    # Kafka producers/consumers
      ├── security/     # JWT filters, authentication
      └── service/       # Infrastructure services (Redis)
    └── presentation/
      ├── controller/   # REST controllers
      └── dto/           # Request/Response DTOs
```

## **Identity Service:**

Handles user authentication and authorization with JWT tokens.

<b>Endpoint</b>	<b>Method</b>	<b>Description</b>
/auth/register	POST	User registration with validation
/auth/login	POST	Authentication, returns JWT token
/auth/logout	POST	Token invalidation
/auth/profile	GET	Retrieve current user profile

## **Fleet Service:**

Manages bus companies, vehicles, and seat configurations.

<b>Endpoint</b>	<b>Method</b>	<b>Description</b>
/bus-companies	GET/POST	List/Create bus companies
/bus-companies/:id	GET/PUT/DELETE	CRUD operations on specific company
/vehicles	GET/POST	List/Create vehicles with company association
/seats/vehicle/:vehicleId	GET	Get all seats for a vehicle

## Route Service:

Handles geographic routing between stations.

Endpoint	Method	Description
/stations	GET/POST	List/Create stations with location data
/routes	GET/POST	List/Create routes with departure/arrival stations

## Schedule Service:

Manages vehicle departure schedules.

Endpoint	Method	Description
/vehicle-schedules	GET/POST	List/Create schedules
/vehicle-schedules?routeId=X	GET	Filter schedules by route

## Ticket Service:

Orchestrates the booking lifecycle with Redis locking and Kafka events.

Endpoint	Method	Description
/tickets	POST	Create ticket (acquires Redis lock)
/tickets/user/me	GET	Get current user's tickets
/tickets/:id	PUT	Update ticket (cancel, etc.)
/tickets/webhook/sepay	POST	Receive SePay payment webhook

### 1.3 Inter-Service Communication

The system employs both synchronous and asynchronous communication patterns:

Synchronous Communication (REST via API Gateway):

All client requests pass through the API Gateway, which routes to appropriate backend services:

Route Pattern	Target Service
/api/auth/**	Identity Service (8081)
/api/bus-companies/**	Fleet Service (8082)
/api/vehicles/**	Fleet Service (8082)
/api/seats/**	Fleet Service (8082)
/api/routes/**	Route Service (8083)
/api/stations/**	Route Service (8083)
/api/vehicle-schedules/**	Schedule Service (8084)
/api/tickets/**	Ticket Service (8085)

## Asynchronous Communication (Kafka Events):

Topic	Producer	Consumer	Event Purpose
<b>payment-events</b>	Ticket Service	Ticket Service	Payment status changes trigger ticket confirmation
<b>ticket-events</b>	Ticket Service	Fleet Service	Ticket booking/cancellation updates seat status
<b>notification-events</b>	Ticket Service	(Notification Service)	User notifications for booking confirmation

## Redis Distributed Locking Flow:

The seat locking mechanism prevents double-booking race conditions:

Step	Action	Redis Operation
1	User selects seat	SETNX seat_lock:scheduleId:seatId LOCKED EX 300
2	Lock acquired	Returns true if key didn't exist
3	Lock denied	Returns false if seat already locked
4	Payment success	DEL seat_lock:scheduleId:seatId
5	TTL expiration	Lock auto-releases after 5 minutes if payment abandoned

## 2. Testing & Verification

The following verification scenarios demonstrate the system's core functionality and architectural goals.

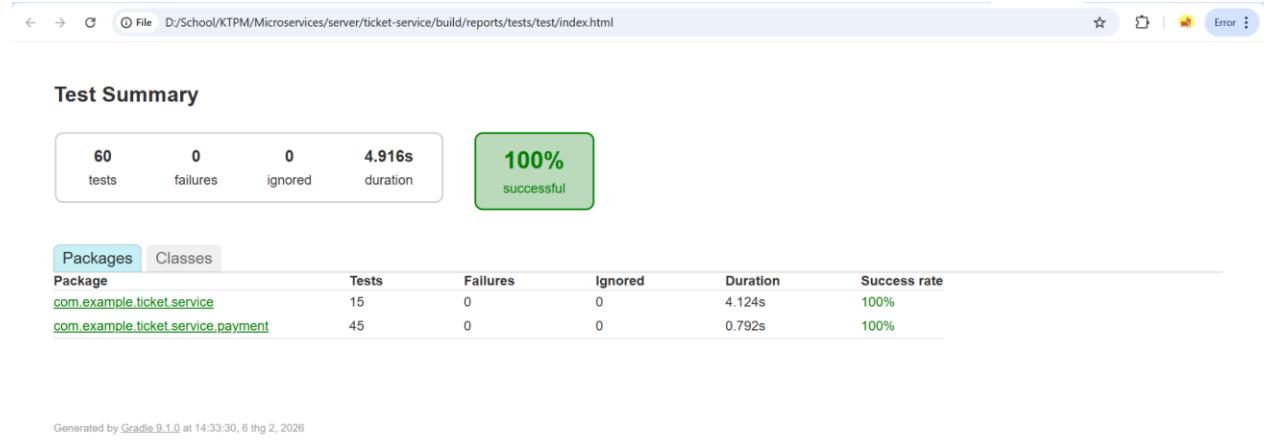


Figure 3. Unit Test Booking\_1

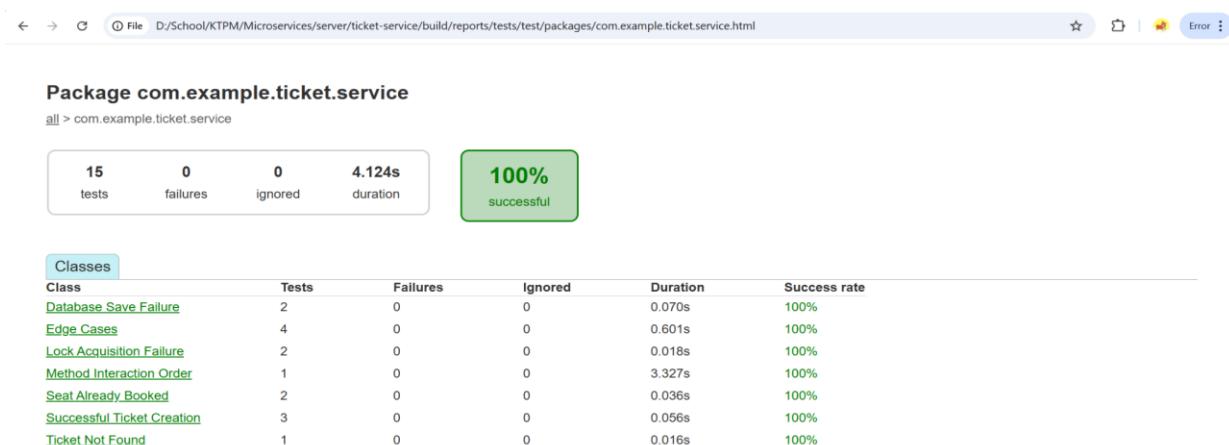


Figure 4. Unit Test Booking\_2

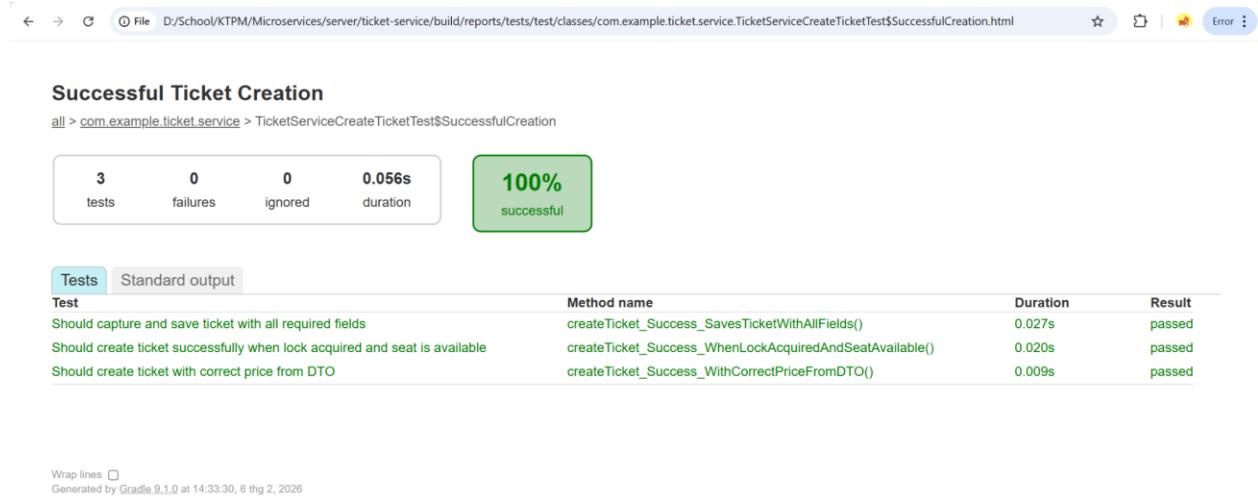


Figure 5. Unit test Booking\_3

```
cp, 0.0.0.0:5672->5672/tcp, ::::5672->5672/tcp, 15671/tcp, 15691-15692/tcp, 25672/tcp, 0.0.0.0:15672->15672/tcp, ::::15672->15672/tcp    rabbitmq
2acb6de6c100    redis:7-alpine          "docker-entrypoint.s..."   36 hours ago   Up 5 minutes   0.0.0.0:6379->6379/tcp
redis
root@Admin-PC:/mnt/d/School/KTPM/Microservices# docker exec -it kafka kafka-cli
OCI runtime exec failed: exec failed: unable to start container process: exec: "kafka-cli": executable file not found in $PATH: unknown
root@Admin-PC:/mnt/d/School/KTPM/Microservices# docker exec -it kafka bash
[appuser@87408f1a7a33 ~]$ kafka-console-consumer \
> --bootstrap-server localhost:9092 \
> --topic ticket.events \
> --from-beginning
{"ticketId":4,"seatId":28,"scheduleId":1,"userId":1,"status":"BOOKED","eventType":"TICKET_BOOKED"}
```

Figure 6. Booking Note Test\_1

```

INFO[0001] [VU 4] User 1 - Response Time: 1611ms      source=console
INFO[0001] [VU 4] User 1 - Response Body: {"success":false,"message":"Seat is currently locked by another user. Please try again later.","responseObject":null} source=console
INFO[0001] [VU 4] ❌ User 1 - Booking FAILED: Seat is currently locked by another user. Please try again later. source=console
INFO[0001] [VU 1] ❌ User 1 - Booking FAILED: Seat is currently locked by another user. Please try again later. source=console
INFO[0001] [VU 2] ❌ User 2 - Booking FAILED: not found ticket with seat id38 source=console
INFO[0121] Load test completed                      source=console
INFO[0121] Test started at: 2026-02-06T15:48:39.466Z   source=console
INFO[0121] Test ended at: 2026-02-06T15:50:41.325Z    source=console
=====
LOAD TEST SUMMARY - GET USER TICKETS
=====

Total Requests: 221
Failed Requests: 1.81%
Avg Response Time: 57.69ms
95th Percentile: 116.32ms
99th Percentile: N/Ams
Tickets Fetched: 217
=====

running (2m01.9s), 00/10 VUs, 220 complete and 0 interrupted iterations
concurrent_booking [=====] 3 VUs      01.8s/30s  3/3 shared iters
smoke      [=====] 1 VUs      30s
load       [=====] 00/10 VUs  50s
ERROR[0122] thresholds on metrics 'http_req_duration, http_req_failed' have been crossed
D:\School\KTPM\Microservices\server\load-tests\ticket-service>

```

Figure 7. Booking Note Test\_2

```

bash: q: command not found
[appuser@87408f1a7a33 ~]$ /q
bash: /q: No such file or directory
[appuser@87408f1a7a33 ~]$ \q
bash: q: command not found
[appuser@87408f1a7a33 ~]$ /q
bash: /q: No such file or directory
[appuser@87408f1a7a33 ~]$ exit
exit
There are stopped jobs.
[appuser@87408f1a7a33 ~]$ read escape sequence
root@Admin-PC:/mnt/d/School/KTPM/Microservices# docker exec -it redis redis-cli
127.0.0.1:6379> MONITOR
OK
1770392920.895563 [0 172.18.0.1:44260] "HELLO" "3"
1770392920.948745 [0 172.18.0.1:44260] "CLIENT" "SETINFO" "lib-name" "Lettuce"
1770392920.948918 [0 172.18.0.1:44260] "CLIENT" "SETINFO" "lib-ver" "6.6.0.RELEASE/643bd47"
1770392921.048673 [0 172.18.0.1:44260] "SET" "seat_lock:1:38" "LOCKED" "EX" "300" "NX"
1770392921.050658 [0 172.18.0.1:44260] "SET" "seat_lock:1:38" "LOCKED" "EX" "300" "NX"
1770392921.050696 [0 172.18.0.1:44260] "SET" "seat_lock:1:38" "LOCKED" "EX" "300" "NX"

```

Figure 8. Booking Note Test\_3

```

C:\Windows\System32\cmd.exe
scenarios: (100.00%) 2 scenarios, 20 max VUs, 1m35s max duration (incl. graceful stop):
  * smoke: 1 looping VUs for 30s (gracefulStop: 5s)
  * load: Up to 20 looping VUs for 50s over 5 stages (gracefulRampDown: 30s, startTime: 35s, gracefulStop: 1
0s)

INFO[0000] Starting load test for Ticket Service - Get User Tickets source=console
INFO[0000] Target URL: http://localhost:8080/api/tickets/user/me source=console
WARN[0000] Health check returned status 404 source=console
INFO[0087] Load test completed source=console
INFO[0087] Test started at: 2026-02-06T15:30:43.636Z source=console
INFO[0087] Test ended at: 2026-02-06T15:32:10.800Z source=console

=====
LOAD TEST SUMMARY - GET USER TICKETS
=====

Total Requests: 310
Failed Requests: 0.32%
Avg Response Time: 89.80ms
95th Percentile: 162.37ms
99th Percentile: N/Ams
Tickets Fetched: 309

=====
running (1m27.2s), 00/20 VUs, 309 complete and 0 interrupted iterations
smoke [=====] 1 VUs 30s
load [=====] 00/20 VUs 50s
D:\School\KTPM\Microservices\server\load-tests\ticket-service>k6 run booking.js

```

Figure 9. Booking Note Test\_4

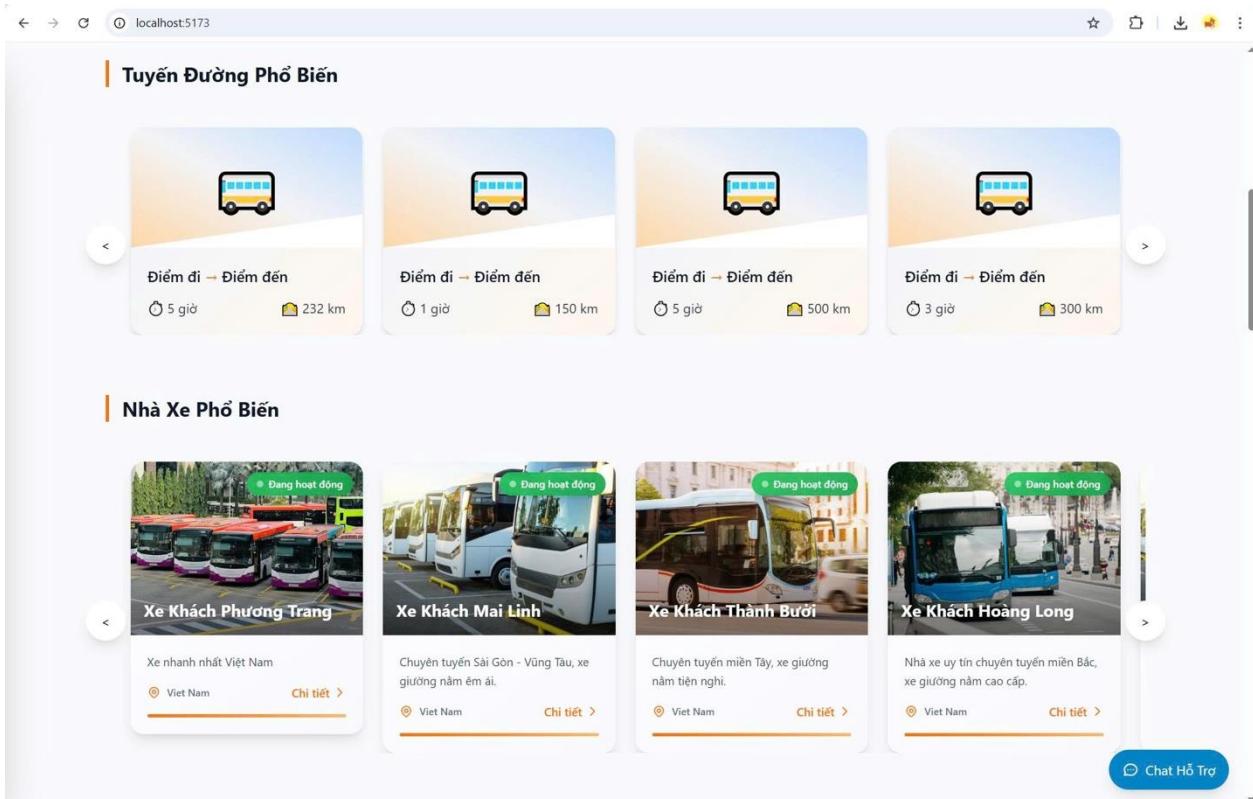


Figure 10. Bus Company List

**Nền Tảng Kết Nối Số 1 Việt Nam**

Chúng tôi tự hào mang đến giải pháp di chuyển thông minh, kết nối hàng triệu hành khách với những chuyến đi an toàn và tiện lợi nhất.



**Mạng Lưới Rộng Khắp**

Hơn 5000+ tuyến đường và 1500+ nhà xe chất lượng cao trên toàn quốc



**Đảm Bảo Có Vé**

Cam kết giữ vé 100%, hoàn ngay 150% nếu có sự cố, giúp bạn an tâm trọn hành trình



**Thanh Toán An Toàn**

Đa dạng phương thức thanh toán bảo mật, tiện lợi và nhanh chóng



**Hỗ Trợ 24/7**

Đội ngũ chuyên viên tận tâm, sẵn sàng giải quyết mọi vấn đề 24/7

ĐỐI TÁC TRUYỀN THÔNG

24h
VTC News
Eva
Afamily
BRVT
Da Nang

VnExpress
Dantri

Chat Hỗ Trợ

**Tin tức nổi bật**

- Xe Limousine – Đẳng cấp thương gia
- Review bến xe Vũng Tàu & Lịch trình
- Top 31 nhà xe limousine đi Đà Lạt

**Về Chúng Tôi**

- Giới Thiệu Kayak
- Tuyển dụng
- Tin tức & Sự kiện

**Tuyến đường phổ biến**

- Sài Gòn - Đà Lạt
- Sài Gòn - Nha Trang
- Sài Gòn - Vũng Tàu
- Hà Nội - Sapa
- Hà Nội - Hạ Long
- Đà Nẵng - Huế

**Đối tác & Bến xe**

- Bến xe Miền Đông
- Bến xe Mỹ Đình
- Nhà xe Phương Trang
- Nhà xe Thành Bưởi
- Nhà xe Kumho Samco

**Hỗ trợ khách hàng**

Chính sách bảo mật

Điều khoản sử dụng

Hướng dẫn đặt vé

**Tổng đài đặt vé:**

**1900 6067**

Hỗ trợ 24/7

**Chứng nhận**

Visa
MC
JCB

© 2026 Kayak Transport Services. All rights reserved.
Facebook
Twitter
Instagram
Chat Hỗ Trợ

Figure 11. Footer

The screenshot shows a search interface for bus routes. On the left, there are four filter sections: 'Điểm khởi hành' (All stations), 'Điểm đến' (All stations), 'Ngày khởi hành' (mm/dd/yyyy), and 'Chọn xe' (All buses). Below these is a blue button labeled 'Áp dụng bộ lọc' (Apply filter). To the right, a summary for 'Phương Trang' route #1 is displayed, showing a thumbnail of a red bus, the route number (Tuyến 1 → Tuyến 2), departure time (12:26 07-02), bus number (Xe #1), route number (Tuyến #1), and a price of 300.000 vnd. There are also 'Chi tiết' (Details) and 'Đặt vé' (Book ticket) buttons.

Figure 12. Bus List

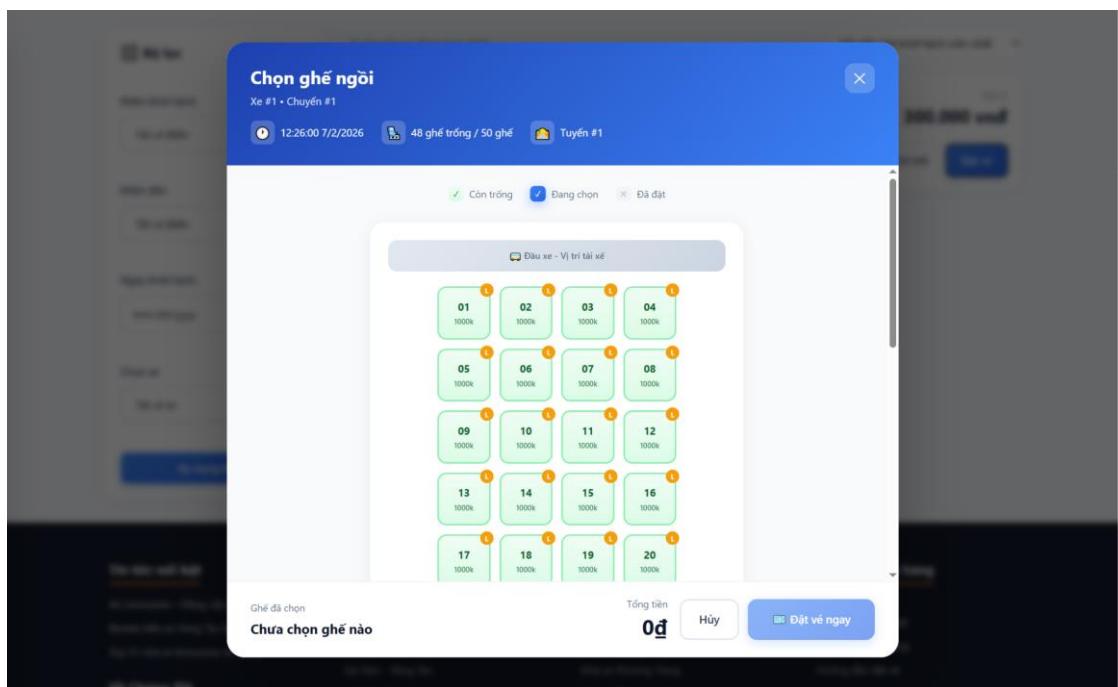


Figure 13. Seat Selection

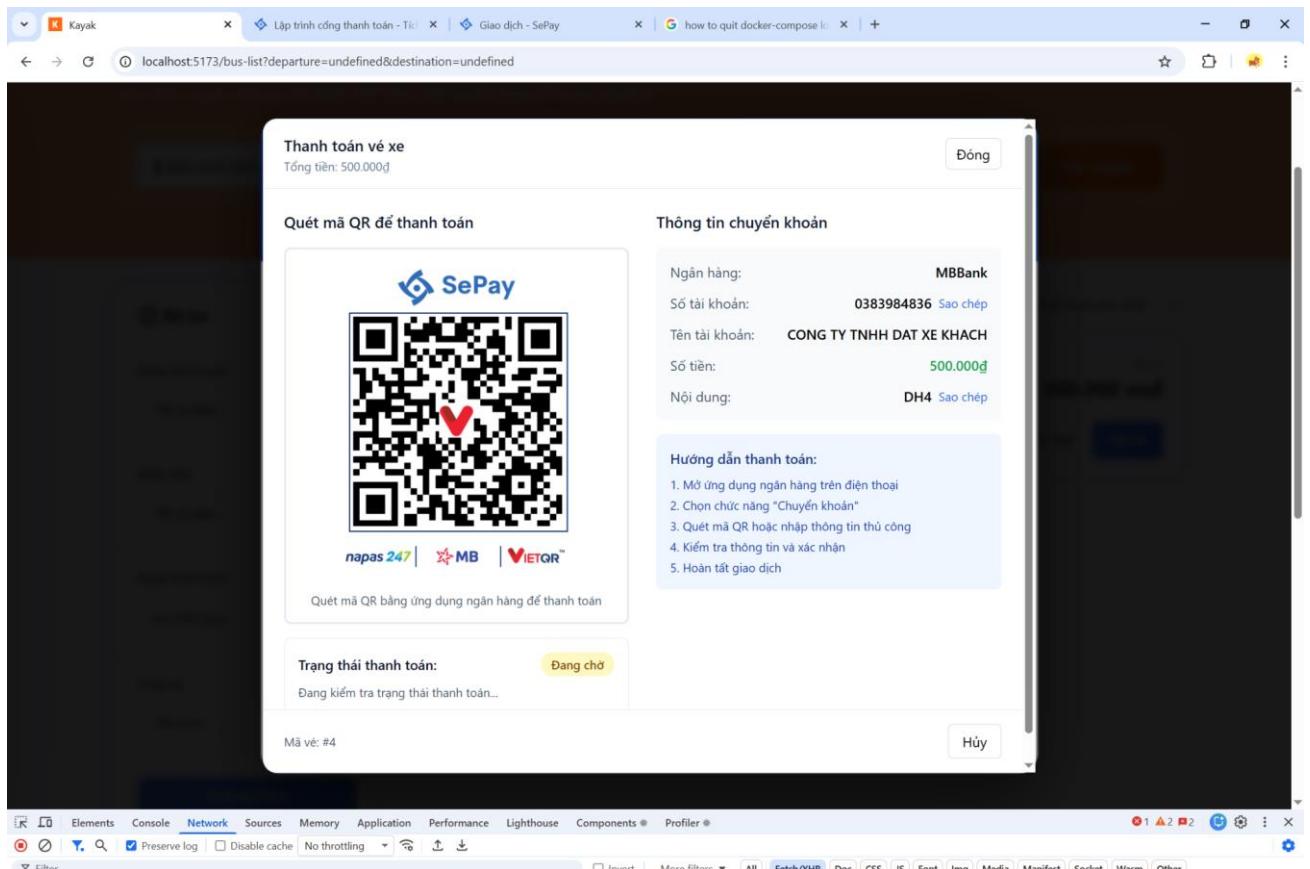


Figure 14. QR Code

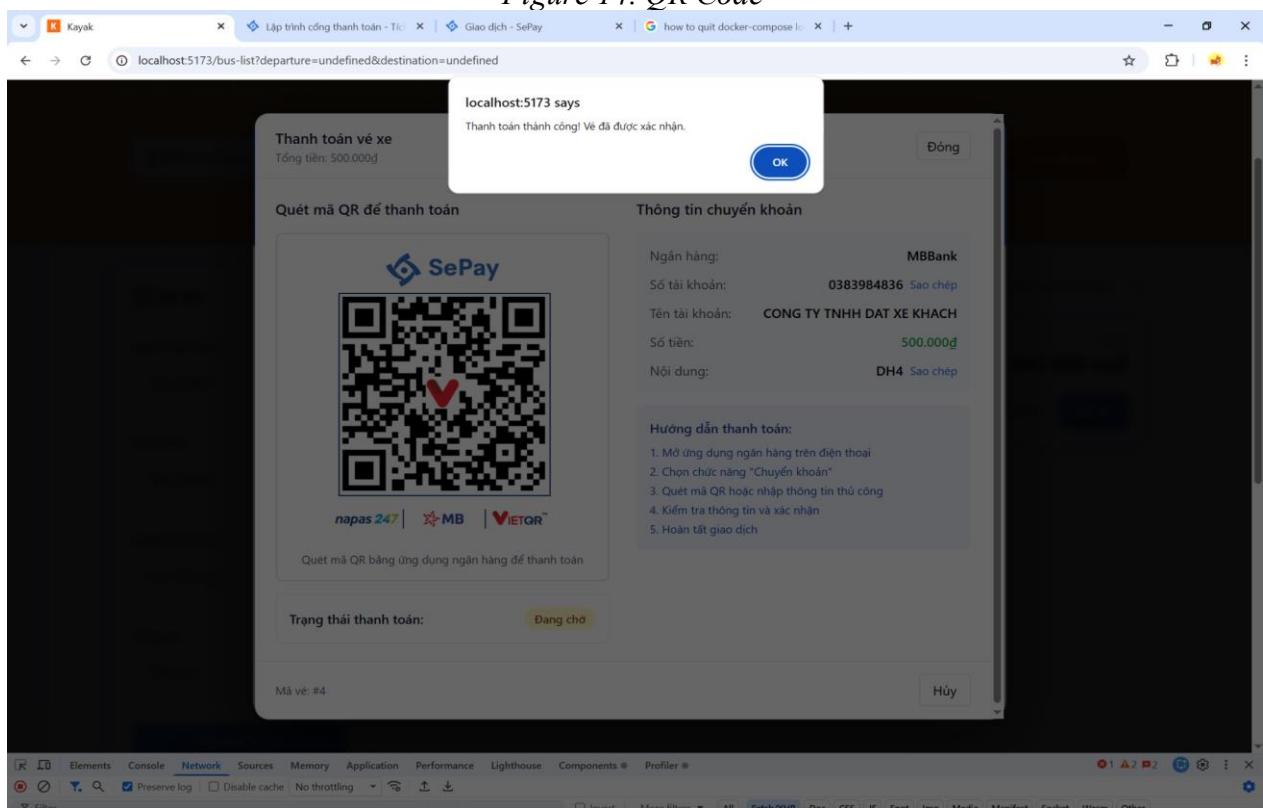


Figure 15. Booking Success

```

  ┌─[n] C:\Program Files\WindowsApps\ngrok.ngrok_3.24.0.0_x64_1g87z0zv29zzc\ngrok.exe - ngrok http 8080
  └─(Ctrl+C to quit)

ngrok

Session Status          online
Account                 lobaoduy2017@gmail.com (Plan: Free)
Update                  update available (version 3.36.0, Ctrl-U to update)
Version                3.24.0-msix
Region                 Asia Pacific (ap)
Latency                56ms
Web Interface          http://127.0.0.1:4040
Forwarding             https://948b-118-70-184-199.ngrok-free.app -> http://localhost:8080

Connections            ttl     opn     rt1     rt5     p50     p90
                       11      0      0.00    0.00   90.18   90.69

HTTP Requests
-----
10:41:44.792 +07 POST /api/tickets/webhook/sepay 200 OK
10:23:33.869 +07 POST /api/tickets/webhook/sepay 200 OK
10:13:52.089 +07 POST /api/tickets/webhook/sepay 200 OK
10:12:00.198 +07 POST /api/tickets/webhook/sepay 200 OK
10:10:01.330 +07 POST /api/tickets/webhook/sepay 200 OK
10:09:09.300 +07 POST /api/tickets/webhook/sepay 200 OK
10:04:30.664 +07 POST /api/tickets/webhook/sepay 200 OK
10:01:49.135 +07 POST /api/tickets/webhook/sepay 200 OK

```

Figure 16. Nginx

```

[1] Select @87408f1a7a33:~
kafka | [2026-02-05 04:30:55,067] TRACE [Controller id=1] Leader imbalance ratio for broker 1 is 0.0 (kafka.controller.KafkaController)
kafka | [2026-02-05 04:35:20,262] INFO [Controller id=1] Processing automatic preferred replica leader election (kafka.controller.KafkaController)
kafka | [2026-02-05 04:35:20,262] TRACE [Controller id=1] Checking need to trigger auto leader balancing (kafka.controller.KafkaController)
kafka | [2026-02-05 04:35:20,263] DEBUG [Controller id=1] Topics not in preferred replica for broker 1 HashMap() (kafka.controller.KafkaController)
kafka | [2026-02-05 04:35:20,263] TRACE [Controller id=1] Leader imbalance ratio for broker 1 is 0.0 (kafka.controller.KafkaController)
kafka | [2026-02-05 04:40:04,841] INFO [Controller id=1] Processing automatic preferred replica leader election (kafka.controller.KafkaController)
kafka | [2026-02-05 04:40:04,841] TRACE [Controller id=1] Checking need to trigger auto leader balancing (kafka.controller.KafkaController)
kafka | [2026-02-05 04:40:04,842] DEBUG [Controller id=1] Topics not in preferred replica for broker 1 HashMap() (kafka.controller.KafkaController)
kafka | [2026-02-05 04:40:04,842] TRACE [Controller id=1] Leader imbalance ratio for broker 1 is 0.0 (kafka.controller.KafkaController)
kafka-console-consumer.sh \
--bootstrap-server localhost:9092 \
--topic ticket.events \
^Z
[2]+  Stopped                  docker-compose logs -f kafka
root@Admin-PC:/mnt/d/School/KTPM/Microservices# kafka-console-consumer.sh \
> --bootstrap-server localhost:9092 \
> --topic ticket.events \
> --from-beginning
kafka-console-consumer.sh: command not found
root@Admin-PC:/mnt/d/School/KTPM/Microservices# docker exec -it kafka bash
[appuser@87408f1a7a33 ~]$ ^C
[appuser@87408f1a7a33 ~]$ kafka-console-consumer \
> --bootstrap-server localhost:9092 \
> --topic ticket.events \
> --from-beginning
{"ticketId":4,"seatId":28,"scheduleId":1,"userId":1,"status":"BOOKED","eventType":"TICKET_BOOKED"}■

```

Figure 17. Kafka

Webhooks đã bán					X
Lần bán	Thuộc WebHooks	Phương thức	Kết nối mạng	HTTP Status Code	Lúc
1	#15393 Webhooks <a href="#">Chi tiết &gt;</a>	POST	Thành công	200 ↗ Gọi lại	2026-02-05 10:23:33 35 giây trước
<a href="#">Xem tất cả &gt;</a>					
<a href="#">Đóng</a>					

Figure 18. Webhooks

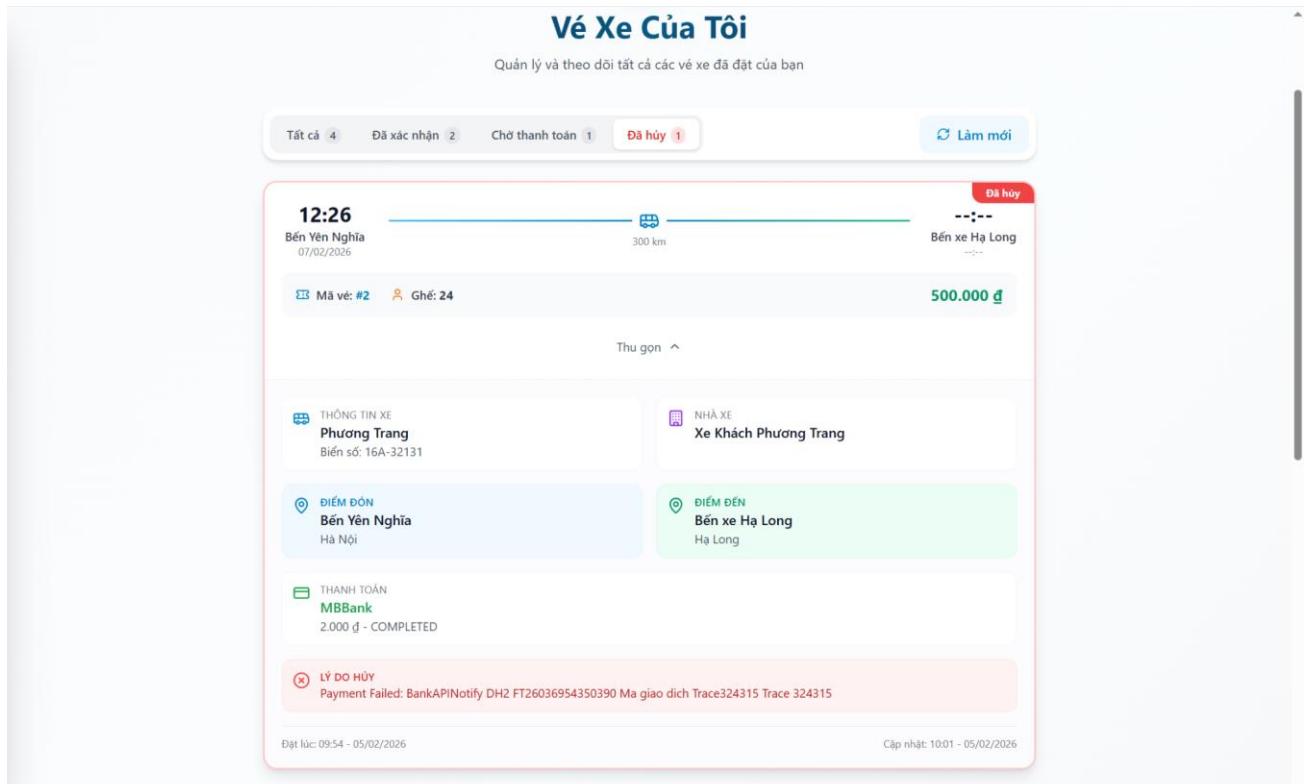


Figure 19. My Tickets\_1

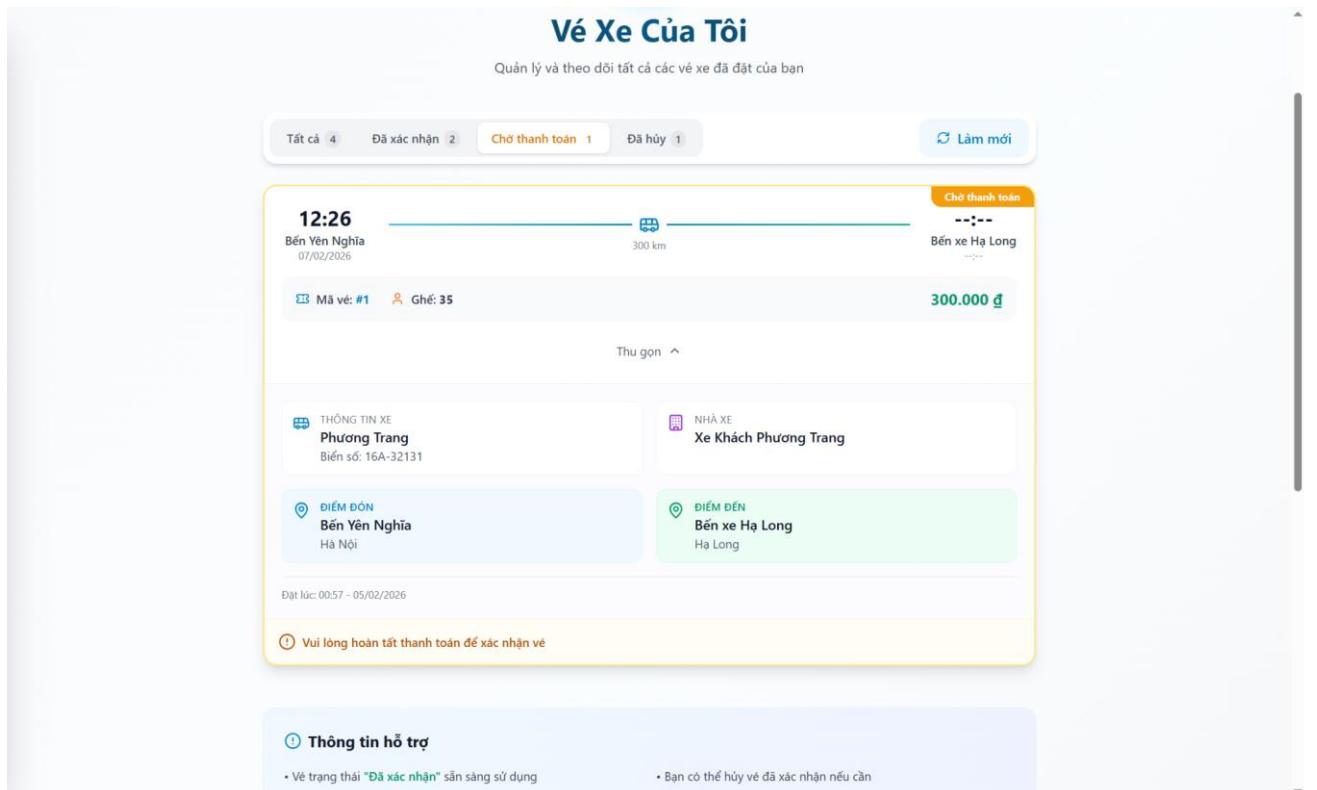


Figure 20. My Tickets\_2

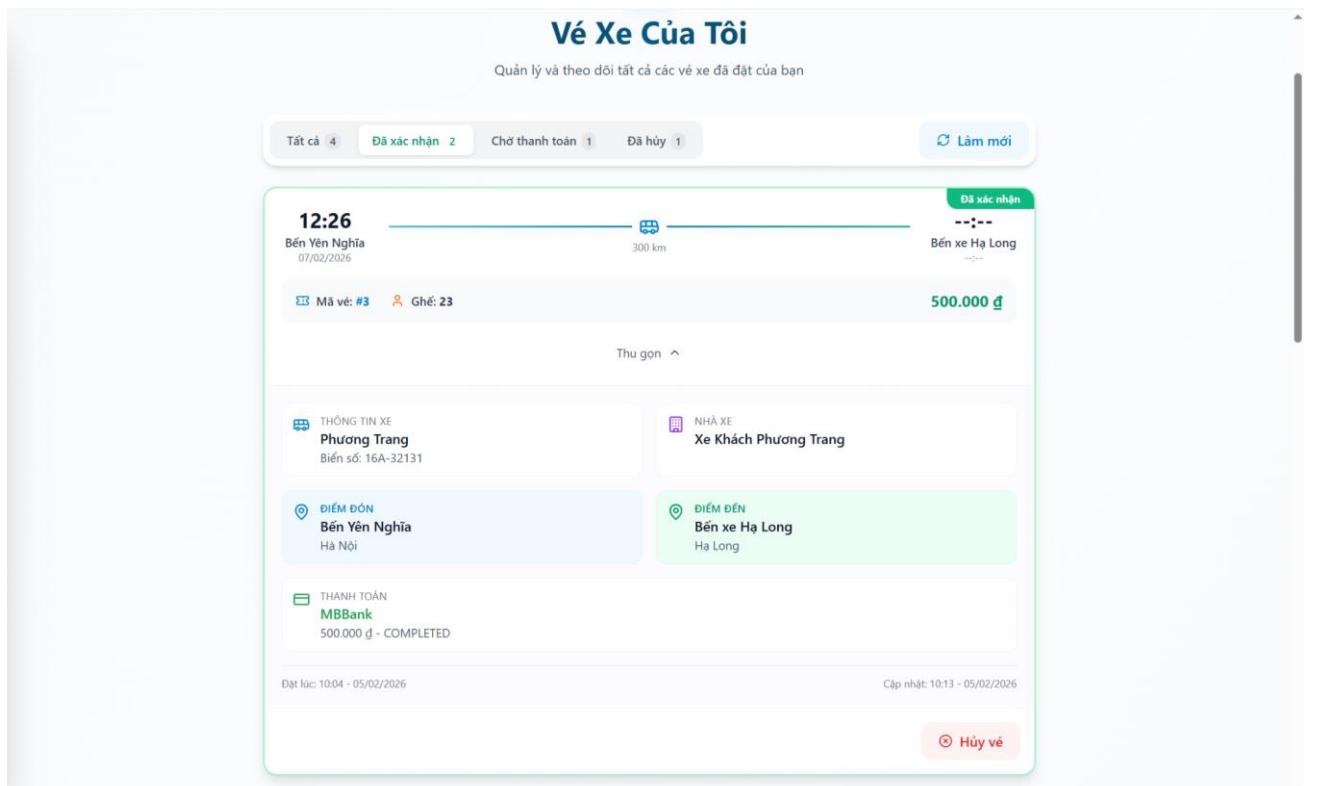


Figure 21. My Tickets\_3

```

Microsoft Windows [Version 10.0.19045.2486]
(c) Microsoft Corporation. All rights reserved.

D:\[School]\KTPM\Microservices>ls
root@Admin-PC:/mnt/d/School/KTPM/Microservices# docker-compose up -d
Creating network "microservices_microservices-network" with driver "bridge"
Creating zookeeper ... done
Creating redis ... done
Creating rabbitmq ... done
Creating kafka ... done
root@Admin-PC:/mnt/d/School/KTPM/Microservices# docker ps
CONTAINER ID IMAGE COMMAND NAMES CREATED STATUS PORTS
87408f1a7a33 confluentinc/cp-kafka:7.5.0 "/etc/confluent/docker... kafka 6 seconds ago Up 5 seconds 0.0.0.0:9092->9092/tcp, :::9092->9092/tcp, 0.0.0.0:29092->29092/tcp, :::29092->29092/tcp
954f174511ca confluentinc/cp-zookeeper:7.5.0 "/etc/confluent/docker... zookeeper 7 seconds ago Up 6 seconds 2888/tcp, 0.0.0.0:2181->2181/tcp, :::2181->2181/tcp, 3888/tcp
6400cb23804c2 rabbitmq:3-management-alpine "/docker-entrypoint.s... rabbitmq 7 seconds ago Up 6 seconds 4369/tcp, 5671/tcp, 0.0.0.0:5672->5672/tcp, :::5672->5672/tcp, 15671/tcp, 15691-15692/tcp
2acbd6e6c100 redis:7-alpine "/docker-entrypoint.s... redis 7 seconds ago Up 6 seconds 0.0.0.0:6379->6379/tcp, :::6379->6379/tcp
aec362098ede vanchuyenvnhtancom_socket-server "/docker-entrypoint.s... vanchuyenvnhtancom_socket-server_1 5 months ago Up About a minute 0.0.0.0:5000->5000/tcp, :::5000->5000/tcp
root@Admin-PC:/mnt/d/School/KTPM/Microservices# docker ps
CONTAINER ID IMAGE COMMAND NAMES CREATED STATUS PORTS
87408f1a7a33 confluentinc/cp-kafka:7.5.0 "/etc/confluent/docker... kafka 6 minutes ago Up 6 minutes 0.0.0.0:9092->9092/tcp, :::9092->9092/tcp, 0.0.0.0:29092->29092/tcp, :::29092->29092/tcp
954f174511ca confluentinc/cp-zookeeper:7.5.0 "/etc/confluent/docker... zookeeper 6 minutes ago Up 6 minutes 2888/tcp, 0.0.0.0:2181->2181/tcp, :::2181->2181/tcp, 3888/tcp

```

Figure 22. Docker Containers

### 3. Conclusion & Reflection

This project successfully implemented a comprehensive Bus Ticket Booking System using a Microservices Architecture with Event-Driven communication patterns. The architecture demonstrates several key software engineering principles:

#### Achievements:

Goal	Achievement
<b>Microservices Decomposition</b>	6 independent services with clear bounded contexts
<b>Event-Driven Architecture</b>	Kafka-based async messaging for payment and ticket events
<b>Distributed Locking</b>	Redis-based seat locking with graceful degradation
<b>API Gateway Pattern</b>	Centralized routing, authentication, and CORS handling

<b>Clean Architecture</b>	Layered service structure with separation of concerns
<b>Database Per Service</b>	Independent MySQL databases per service domain

### 3.1 Lessons Learned

Microservices Complexity Trade-off: Decomposing the application into services introduces operational complexity (deployment, monitoring, distributed transactions). However, the benefits of independent scaling and deployment outweigh the costs for this use case.

Event-Driven Resilience: Kafka provides excellent decoupling between services. The payment → ticket confirmation flow demonstrates how services can operate independently without synchronous dependencies.

Redis for Distributed State: Using Redis for seat locking proved essential for handling concurrent bookings. The TTL-based expiration mechanism prevents orphaned locks from abandoned sessions.

API Gateway as Security Perimeter: Centralizing JWT validation at the gateway simplifies individual service implementations and ensures consistent security enforcement.

Graceful Degradation: Implementing fallback mechanisms (e.g., Redis lock service accepting requests when Redis is unavailable) improves system resilience.

### 3.2 Future Improvements

Improvement	Description	Priority
<b>Service Discovery</b>	Implement Eureka or Consul for dynamic service registration and discovery instead of hardcoded URLs	High
<b>Circuit Breaker</b>	Add Resilience4j circuit breakers for fault tolerance in inter-service calls	High
<b>Centralized Logging</b>	Implement ELK Stack (Elasticsearch, Logstash, Kibana) for aggregated log analysis	Medium
<b>Distributed Tracing</b>	Add Zipkin/Jaeger for request tracing across services	Medium

<b>Kubernetes Deployment</b>	Containerize and deploy to Kubernetes for production-grade orchestration	High
<b>Notification Service</b>	Implement dedicated microservice for email/SMS notifications via RabbitMQ	Medium
<b>Review Service</b>	Add customer review and rating functionality for bus companies	Low
<b>Chatbot Integration</b>	Implement AI-powered chatbot for customer support	Low
<b>Payment Retry</b>	Add retry mechanism for failed webhook deliveries	Medium
<b>Caching Layer</b>	Implement Redis caching for frequently accessed data (routes, schedules)	Medium

## REFERENCES

- [1] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, 4th ed. Boston, MA, USA: Addison-Wesley, 2021.
- [2] M. Richards and N. Ford, *Fundamentals of Software Architecture: An Engineering Approach*. Sebastopol, CA, USA: O'Reilly Media, 2020.
- [3] S. Newman, *Building Microservices: Designing Fine-Grained Systems*, 2nd ed. Sebastopol, CA, USA: O'Reilly Media, 2021.
- [4] C. Richardson, *Microservices Patterns: With Examples in Java*. Shelter Island, NY, USA: Manning Publications, 2018.
- [5] G. Hohpe and B. Woolf, *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Boston, MA, USA: Addison-Wesley, 2004.
- [6] M. Fowler, “Microservices Architecture,” [martinfowler.com](https://martinfowler.com), 2014. [Online]. Available: <https://martinfowler.com/microservices/architecture.html>
- [7] S. Newman, “API Gateway Pattern,” [microservices.io](https://microservices.io). [Online]. Available: <https://microservices.io/patterns/apigateway.html>
- [8] Apache Software Foundation, “Apache Kafka Documentation,” 2025. [Online]. Available: <https://kafka.apache.org/documentation/>
- [9] Redis Ltd., “Redis Documentation – Distributed Locks and Caching,” 2025. [Online]. Available: <https://redis.io/docs/>

- [10] S. Brown, “The C4 Model for Visualising Software Architecture,” 2023. [Online]. Available: <https://c4model.com/>
- [11] OWASP Foundation, “JSON Web Token (JWT) Security Best Practices,” 2024. [Online]. Available: <https://owasp.org/>