Parking Management System - Testing Guide

Prerequisites

- Postman installed
- Java 17+ installed
- Maven installed

Quick Start

1. Start the Application



hash

cd parking-systemmvn clean installmvn spring-boot:run

 $lue{}$ Wait for: Started ParkingSystemApplication on port 8080

2. Import Postman Collection

- 1. Open Postman
- 2. Click **Import** \rightarrow **Raw text**
- 3. Paste the JSON collection (provided separately)
- 4. Click Import

Testing Flow (5 Minutes)

Phase 1: Setup Data

1.1 Create Parking Slots



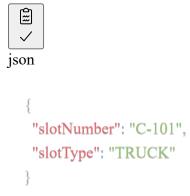
POST http://localhost:8080/api/slots Content-Type: application/json

Create 4 slots (run each separately):

Slot 1 (CAR):

```
json
   "slotNumber": "A-101",
   "slotType": "CAR"
Slot 2 (CAR):
json
   "slotNumber": "A-102",
   "slotType": "CAR"
Slot 3 (BIKE):
json
   "slotNumber": "B-101",
   "slotType": "BIKE"
```

Slot 4 (TRUCK):



Expected: Each returns 201 Created with slot details

1.2 Register Vehicles



POST http://localhost:8080/api/vehicles Content-Type: application/json

Vehicle 1 (CAR):

```
json

{
    "licensePlate": "MH01AB1234",
    "ownerName": "John Doe",
    "vehicleType": "CAR"
```

Vehicle 2 (BIKE):

```
json

{
    "licensePlate": "MH02CD5678",
    "ownerName": "Jane Smith",
    "vehicleType": "BIKE"
}
```

Vehicle 3 (TRUCK):



Expected: Each returns 201 Created

▲ COPY the id from each response - you'll need it!

Phase 2: Core Operations

2.1 List All Vehicles



GET http://localhost:8080/api/vehicles

Expected: 200 OK with list of 3 vehicles

2.2 List All Slots



GET http://localhost:8080/api/slots

Expected: 200 OK with list of 4 slots (all available)

2.3 Park a Vehicle



POST http://localhost:8080/api/park Content-Type: application/json

Body (replace with actual vehicle ID):



json

```
{
    "vehicleId": "PASTE_VEHICLE_ID_HERE"
}
```

Expected: 201 Created with ticket details



Sample Response:



json

```
"id": "ticket-123-abc",

"vehicle": {

"id": "vehicle-456",

"licensePlate": "MH01AB1234",

"vehicleType": "CAR"

},

"slot": {

"slotNumber": "A-101",

"isAvailable": false

},

"entryTime": "2024-10-17T11:30:00",

"exitTime": null
```

2.4 Check Available Slots



GET http://localhost:8080/api/slots?available=true

Expected: 200 OK - Should show 3 slots (one is occupied)

2.5 Get Ticket Details



GET http://localhost:8080/api/tickets/{ticketId}

Replace {ticketId} with actual ticket ID

Expected: 200 OK with full ticket details

2.6 Unpark Vehicle



POST http://localhost:8080/api/unpark/{ticketId}

Replace {ticketId} with actual ticket ID

Expected: 200 OK with updated ticket (exitTime populated)

2.7 Verify Slot is Free Again



GET http://localhost:8080/api/slots?available=true

Expected: 200 Οκ - Should show 4 slots (all available again)

Phase 3: Error Scenario Testing

3.1 Duplicate License Plate



POST http://localhost:8080/api/vehicles

```
| "licensePlate": "MH01AB1234",
    "ownerName": "Another Person",
    "vehicleType": "CAR"
    }
```

X Expected: 400 Bad Request

```
son

{
    "status": 400,
    "message": "Vehicle with this license plate already exists"
}
```

3.2 Park Already Parked Vehicle

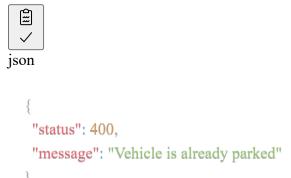


POST http://localhost:8080/api/park

```
json

{
    "vehicleId": "ALREADY_PARKED_VEHICLE_ID"
}
```

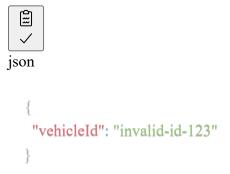
X Expected: 400 Bad Request



3.3 Park with Invalid Vehicle ID



POST http://localhost:8080/api/park



X Expected: 404 Not Found



```
ison
{
    "status": 404,
    "message": "Vehicle not found"
}
```

3.4 No Available Slots

Setup:

- 1. Create only 1 CAR slot
- 2. Register 2 CAR vehicles
- 3. Park first vehicle
- 4. Try to park second vehicle X





3.5 Validation Errors



POST http://localhost:8080/api/vehicles



```
{
  "licensePlate": "",
  "ownerName": "",
  "vehicleType": null
}
```

X Expected: 400 Bad Request

```
ijson

{
    "status": 400,
    "errors": {
        "licensePlate": "License plate is required",
        "ownerName": "Owner name is required",
        "vehicleType": "Vehicle type is required"
    }
}
```

3.6 Unpark Already Unparked Vehicle



POST http://localhost:8080/api/unpark/{ticketId}

(Use same ticket ID twice)

X Expected: 400 Bad Request

```
json

{
    "status": 400,
    "message": "Vehicle already unparked"
}
```

3.7 Get Non-existent Ticket



GET http://localhost:8080/api/tickets/invalid-ticket-id

X Expected: 404 Not Found



isor

```
{
  "status": 404,
  "message": "Ticket not found"
}
```

Phase 4: Filter Testing

4.1 Filter Slots by Type



GET http://localhost:8080/api/slots?type=CAR
GET http://localhost:8080/api/slots?type=BIKE
GET http://localhost:8080/api/slots?type=TRUCK

Expected: Only slots of specified type

4.2 Filter Slots by Availability



GET http://localhost:8080/api/slots?available=false

Expected: Only occupied slots

4.3 Combined Filters



GET http://localhost:8080/api/slots?available=true&type=CAR

Expected: Only available CAR slots

Complete Test Checklist

Happy Path

- Create parking slots
- Register vehicles

- List all vehicles
- List all slots
- Park a vehicle
- Get ticket details
- Check available slots
- Unpark vehicle
- Verify slot is free

Error Scenarios X

- Duplicate license plate
- Park already parked vehicle
- Invalid vehicle ID
- No available slots
- Missing required fields
- Unpark twice
- Non-existent resources

Filters 🔍

- Filter by slot type
- Filter by availability
- Combined filters

Quick Reference - All Endpoints

Method	Endpoint	Purpose
POST	/api/vehicles	Register vehicle
GET	/api/vehicles	List all vehicles
GET	/api/vehicles/{id}	Get vehicle by ID
POST	/api/slots	Create parking slot
GET	/api/slots	List slots (with filters)
POST	/api/park	Park a vehicle
POST	/api/unpark/{ticketId}	Unpark a vehicle
GET	/api/tickets/{id}	Get ticket details

Pro Tips

Using Postman Environment Variables

After Vehicle Registration: In the Tests tab, add:



javascript

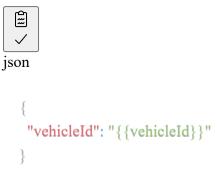
```
var response = pm.response.json();
pm.environment.set("vehicleId", response.id);
```

After Parking:

```
javascript
```

```
var response = pm.response.json();
pm.environment.set("ticketId", response.id);
```

Then use in requests:



Expected Test Duration

Phase	T	ime
<pre>Setup (Slots + Vehicles)</pre>	2	min
Core Operations	2	min
Error Scenarios	3	min
Filter Testing	1	min
Total	8	min

Common Issues & Solutions

Issue 1: Port Already in Use



Error: Port 8080 is already in use

Solution:



bash

```
# Kill process on port 8080

lsof-ti:8080 | xargs kill -9

# Or change port in application.properties
server.port=8081
```

Issue 2: H2 Database Console

Access: http://localhost:8080/h2-console

• JDBC URL: jdbc:h2:mem:parkingdb

Username: sa Password: (blank)

Issue 3: Application Not Starting



hach

Clean and rebuild
mvn clean install
Check Java version
java -version # Should be 17+

Sample Test Data Set

Complete Working Example

Slots:



A-101 (CAR) A-102 (CAR) B-101 (BIKE) B-102 (BIKE) C-101 (TRUCK)

Vehicles:



MH01AB1234 - John Doe (CAR) MH02CD5678 - Jane Smith (BIKE)

MH03EF9012 - Mike Johnson (TRUCK)

MH04GH3456 - Sarah Williams (CAR)

Operations:

- 1. Park MH01AB1234 → Slot A-101
- 2. Park MH02CD5678 → Slot B-101
- 3. Unpark MH01AB1234 \rightarrow Slot A-101 free
- 4. Park MH04GH3456 \rightarrow Slot A-101 (reused)

Success Criteria

- All 8 endpoints working
- All validations triggering correctly
- ✓ All error scenarios handled properly
- Filters working as expected
- Data persisting during application lifecycle
- Proper HTTP status codes returned
- Clean JSON responses

Ready for Demo? 🞉



Follow this checklist before presenting:

- 1. Application starts without errors
- 2. Can create slots
- 3. Can register vehicles
- 4. Can park and unpark
- 5. Z Errors are handled gracefully
- 6. Logs are visible in console
- 7. Postman collection imported

You're good to go! 🚀

