Parking Lot Management System

API Documentation

Version: 1.0.0

Base URL: http://localhost:3000

Table of Contents

Introduction

This document provides detailed information about the Parking Lot Management System API, which allows you to create and manage a parking lot, park and remove cars, and query information about parked cars.

The API is built with NestJS and follows RESTful principles. All requests and responses use JSON format.

Authentication

Currently, the API does not implement authentication mechanisms. All endpoints are publicly accessible.

API Endpoints

1. Parking Lot Management

1.1 Create a Parking Lot

Creates a new parking lot with the specified number of slots.

Request:

- **Method:** POST
- **Endpoint:** `/parking_lot`
- **Content-Type:** application/json

Request Body:

Field	Type	Required	Description
`no_of_slot`	integer	Yes	Number of parking slots to create (minimum: 1)

Example Request:

```json
{
  "no_of_slot": 5
}
```

Success Response:

- **Status Code:** 201 Created
- **Example:**

```json
{
  "total_slot": 5
}
```

Error Response:

- **Status Code:** 400 Bad Request
- **Example:**

```json
{
  "statusCode": 400,
  "message": ["no_of_slot must be a positive integer"],
  "error": "Bad Request"
}
```

1.2 Expand Parking Lot

Increases the size of an existing parking lot by adding more slots.

Request:

- **Method:** PATCH
- **Endpoint:** `/parking_lot`
- **Content-Type:** application/json

Request Body:

Field	Type	Required	Description
`increment_slot`	integer	Yes	Number of additional slots to add

Example Request:

```
json

{
  "increment_slot": 3
}
```

Success Response:

- **Status Code:** 200 OK
- **Example:**

```
json

{
  "total_slots": 8
}
```

Error Response:

- **Status Code:** 400 Bad Request
- **Example:**

```
json

{
  "statusCode": 400,
  "message": ["increment_slot must be a number"],
  "error": "Bad Request"
}
```

2. Car Parking Operations

2.1 Park a Car

Parks a car in the nearest available slot.

Request:

- **Method:** POST
- **Endpoint:** `/park`
- **Content-Type:** application/json

Request Body:

Field	Type	Required	Description
`reg_no`	string	Yes	Car registration number
`color`	string	Yes	Car color

Example Request:

```json
{
  "reg_no": "ABC123",
  "color": "Red"
}
```

Success Response:

- **Status Code:** 201 Created
- **Example:**

```json
{
  "allocated_slot": 1
}
```

Error Responses:

- **Status Code:** 400 Bad Request
  - **If parking lot is not initialized:**

json

```json
{
  "statusCode": 400,
  "message": "Parking lot not initialized. Please create parking lot first.",
  "error": "Bad Request"
}
```

- **If parking lot is full:**

json

```json
{
  "statusCode": 400,
  "message": "Parking is full. No slots available.",
  "error": "Bad Request"
}
```

- **If required fields are missing:**

json

```json
{
  "statusCode": 400,
  "message": ["reg_no must be a string", "color must be a string"],
  "error": "Bad Request"
}
```

2.2 Clear a Parking Slot

Removes a car from a parking slot, making it available again. Can be cleared by either slot number or registration number.

Request:

- **Method:** POST
- **Endpoint:** `/clear`
- **Content-Type:** application/json

Request Body:

Field	Type	Required	Description
`slot_number`	integer	No*	Slot number to clear
`car_registration_no`	string	No*	Registration number of car to remove

*At least one of these fields must be provided

Example Request (by slot number):

```json
{
  "slot_number": 1
}
```

Example Request (by registration number):

```json
{
  "car_registration_no": "ABC123"
}
```

Success Response:

- **Status Code:** 200 OK
- **Example:**

```json
{
  "cleared_slot": 1
}
```

Error Responses:

- **Status Code:** 400 Bad Request
  - **If neither field is provided:**

  ```json
  {
    "statusCode": 400,
    "message": "Please provide either slot_number or car_registration_no",
    "error": "Bad Request"
  }
  ```

- **Status Code:** 404 Not Found
  - **If slot is already free:**

```json
{
  "statusCode": 404,
  "message": "Slot already free.",
  "error": "Not Found"
}
```

- **If car with given registration is not found:**

```json
{
  "statusCode": 404,
  "message": "Car with this registration number not found.",
  "error": "Not Found"
}
```

3. Parking Queries

3.1 Get Parking Status

Retrieves the current status of all occupied parking slots.

Request:

- **Method:** GET
- **Endpoint:** `/status`

Success Response:

- **Status Code:** 200 OK
- **Example:**

```json
[
  {
    "slot_no": 1,
    "car_registration": "ABC123",
    "car_color": "Red"
  },
  {
    "slot_no": 2,
    "car_registration": "XYZ789",
    "car_color": "Blue"
  }
]
```

3.2 Get Registration Numbers by Color

Retrieves all car registration numbers for cars of a specific color.

Request:

- **Method:** GET
- **Endpoint:** `/registration_numbers/color/:color`

URL Parameters:

Parameter	Type	Required	Description
`color`	string	Yes	Color of cars to search for

Example Request:

```
GET /registration_numbers/color/red
```

Success Response:

- **Status Code:** 200 OK
- **Example:**

```
json

{
  "color": "red",
  "registration_numbers": ["ABC123", "DEF456"]
}
```

Error Response:

- **Status Code:** 404 Not Found
- **Example:**

```
json

{
  "statusCode": 404,
  "message": "No cars found with color: red",
  "error": "Not Found"
}
```

3.3 Get Slot Number by Registration Number

Finds the slot number where a car with a specific registration number is parked.

Request:

- **Method:** GET
- **Endpoint:** `/slot`

Query Parameters:

Parameter	Type	Required	Description
`registration_number`	string	Yes	Car registration number to search for

Example Request:

```
GET /slot?registration_number=ABC123
```

Success Response:

- **Status Code:** 200 OK
- **Example:**

```json
{
  "registration_number": "ABC123",
  "slot_number": 1
}
```

Error Responses:

- **Status Code:** 400 Bad Request
  - **If registration number is not provided:**

  ```json
  {
    "statusCode": 400,
    "message": "Registration number is required",
    "error": "Bad Request"
  }
  ```

- **Status Code:** 404 Not Found
  - **If car with given registration number is not found:**

```json
{
  "statusCode": 404,
  "message": "Car with registration number ABC123 not found",
  "error": "Not Found"
}
```

3.4 Get Slot Numbers by Color

Finds all slot numbers where cars of a specific color are parked.

Request:

- **Method:** GET
- **Endpoint:** `/slots/color/:color`

URL Parameters:

Parameter	Type	Required	Description
`color`	string	Yes	Color of cars to search for

Example Request:

```
GET /slots/color/blue
```

Success Response:

- **Status Code:** 200 OK
- **Example:**

```json
{
  "color": "blue",
  "slot_numbers": [2, 4, 6]
}
```

Error Response:

- **Status Code:** 404 Not Found
- **Example:**

```json
{
  "statusCode": 404,
  "message": "No cars found with color: blue",
  "error": "Not Found"
}
```

Data Models

Car

Field	Type	Description
`reg_no`	string	Registration number of the car
`color`	string	Color of the car

Response Codes

Status Code	Description
200	OK - The request has succeeded
201	Created - The request has been fulfilled and resulted in a new resource being created
400	Bad Request - The server could not understand the request due to invalid syntax or missing parameters
404	Not Found - The server cannot find the requested resource
500	Internal Server Error - The server has encountered a situation it doesn't know how to handle

---

Error Handling

All API endpoints return standardized error responses in the following format:

```json
{
  "statusCode": 400,
  "message": "Error message describing what went wrong",
  "error": "Error type (e.g., Bad Request, Not Found)"
}
```

Usage Examples

Complete Flow Example

Here's a complete example of using the API to manage a parking lot:

1. Create a parking lot with 3 slots:

Request:

```
POST /parking_lot
Content-Type: application/json

{
  "no_of_slot": 3
}
```

Response:

```json
{
  "total_slot": 3
}
```

2. Park three cars:

Request 1:

```
POST /park
Content-Type: application/json

{
  "reg_no": "ABC123",
  "color": "Red"
}
```

Response 1:

```json
{
  "allocated_slot": 1
}
```

Request 2:

```
POST /park
Content-Type: application/json

{
  "reg_no": "DEF456",
  "color": "Blue"
}
```

Response 2:

```json
{
  "allocated_slot": 2
}
```

Request 3:

```
POST /park
Content-Type: application/json

{
  "reg_no": "GHI789",
  "color": "Red"
}
```

Response 3:

```json
{
  "allocated_slot": 3
}
```

3. Check status:

Request:

```
GET /status
```

Response:

```json
[
  {
    "slot_no": 1,
    "car_registration": "ABC123",
    "car_color": "Red"
  },
  {
    "slot_no": 2,
    "car_registration": "DEF456",
    "car_color": "Blue"
  },
  {
    "slot_no": 3,
    "car_registration": "GHI789",
    "car_color": "Red"
  }
]
```

4. Find all red cars:

Request:

```
GET /registration_numbers/color/Red
```

Response:

```json
{
  "color": "Red",
  "registration_numbers": ["ABC123", "GHI789"]
}
```

5. Clear a slot:

Request:

```
POST /clear
Content-Type: application/json

{
  "slot_number": 2
}
```

Response:

```json
{
  "cleared_slot": 2
}
```

6. Expand parking lot:

Request:

```
PATCH /parking_lot
Content-Type: application/json

{
  "increment_slot": 2
}
```

Response:

```json
{
  "total_slots": 5
}
```

End of API Documentation