Backend 2.3_CW Exercises

Introduction to POST Requests

ex01: writing a basic POST request

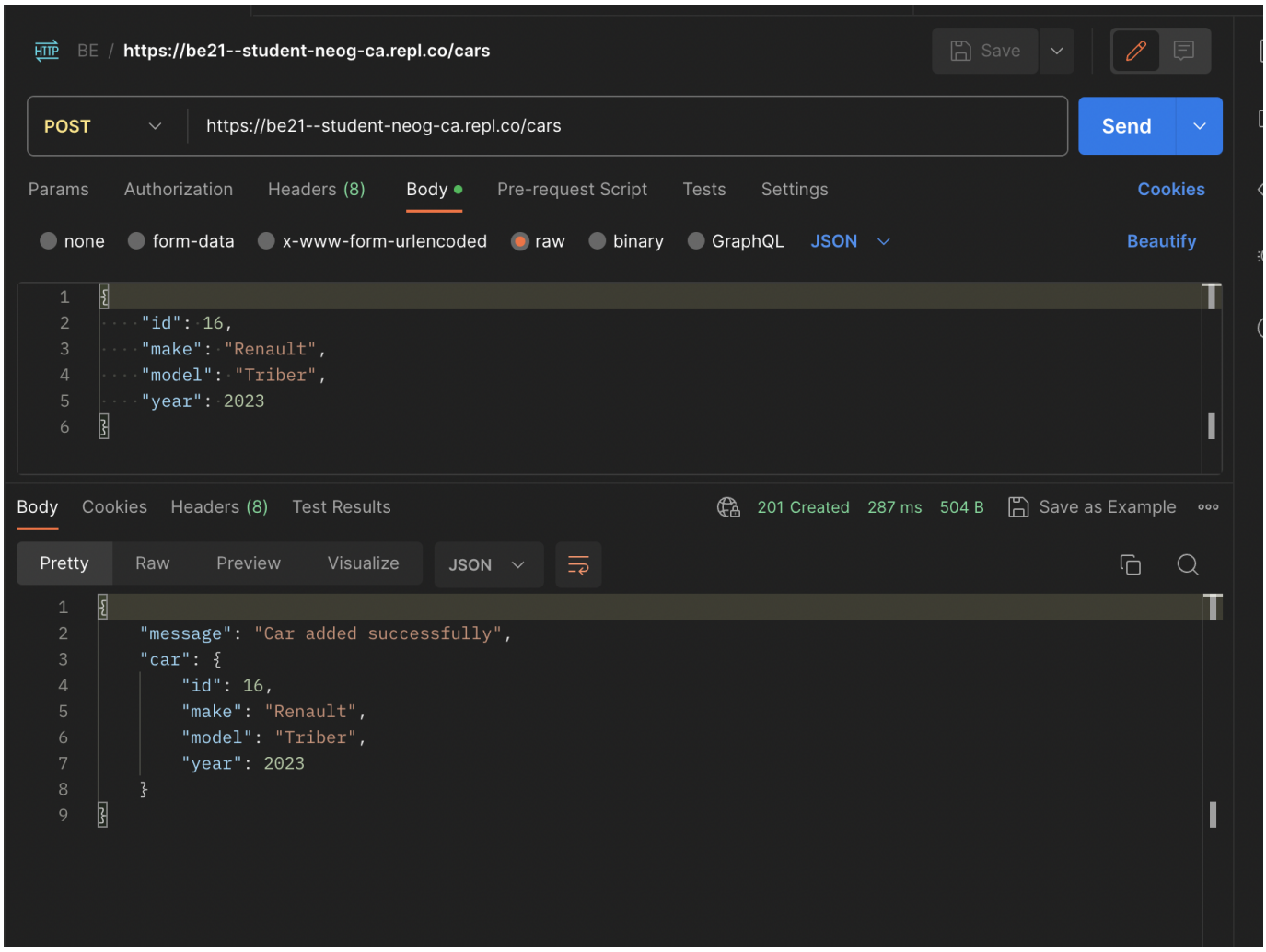
In this exercise, we'll create a simple POST request that allows us to add a new car to our inventory.

challenge

Create POST Route for Adding Cars: Define a POST route at `/cars` that receives JSON data of a new car and adds it to the inventory.

Make sure you use the built-in `express.json()` middleware to parse JSON request bodies.

- Good question




```
app.use(express.json());

Syntax:
app.**post**('/cars', (req, res) => {
  const newCar = req.body; // this is how you can access the data you want to post

        // your code here
});
```

COPY

solution

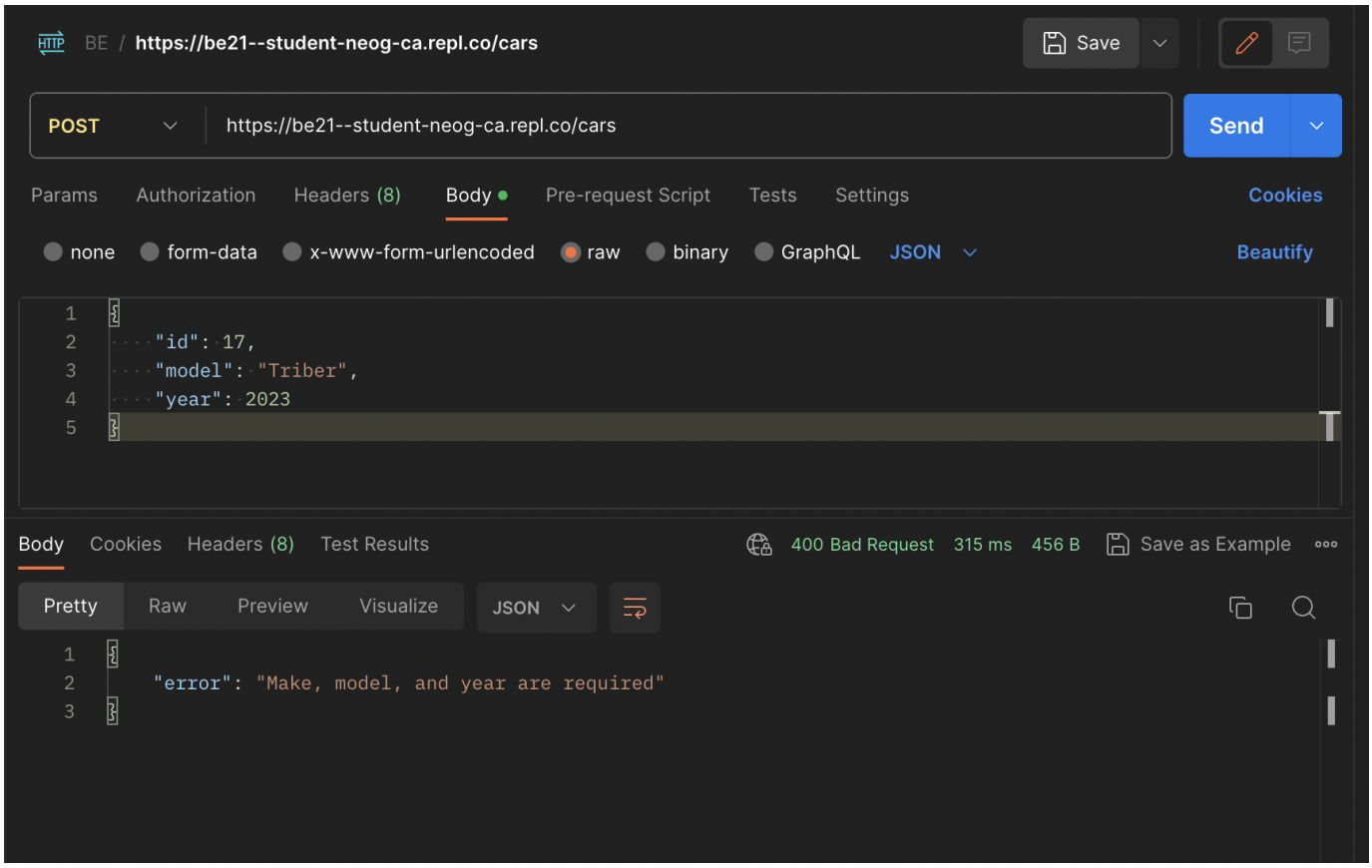
```
app.use(express.json())

app.post('/cars', (req, res) => {
  const newCar = req.body
  cars.push(newCar)
  res.status(201).json({ message: 'Car added successfully', car: newCar })
})
```

COPY

testing

Test with Postman: In Postman, create a POST request to `https://your-replit-app-url.glitch.me/cars`, and in the request body, send JSON data representing a new car. Observe the response and check if the new car is added to the inventory.

- Solution



ex02: understanding the request body and HTTPS importance

Now, let's break down what happened in the previous exercise, focusing on the request body and the importance of using HTTPS.

understanding

1. Request Body: In a POST request, data is sent in the request body. The server decodes this data and processes it based on the route.
2. Security Concerns: Sending data via the request body isn't inherently secure, especially when dealing with sensitive information. HTTPS (SSL/TLS) encryption is essential to secure data during transmission.

ex03: writing a POST request with data validation

In this exercise, we'll enhance our POST request by adding data validation to ensure the incoming data is valid.

challenge

Modify POST Route for Adding Cars: Update the `/cars` route to validate the incoming data before adding the car to the inventory.

solution

```
app.post('/cars', (req, res) => {
  const newCar = req.body

  if (!newCar.make || !newCar.model || !newCar.year) {
    res.status(400).json({ error: 'Make, model, and year are required' })
  } else {
    cars.push(newCar)
    res.status(201).json({ message: 'Car added successfully', car: newCar })
  }
})
```

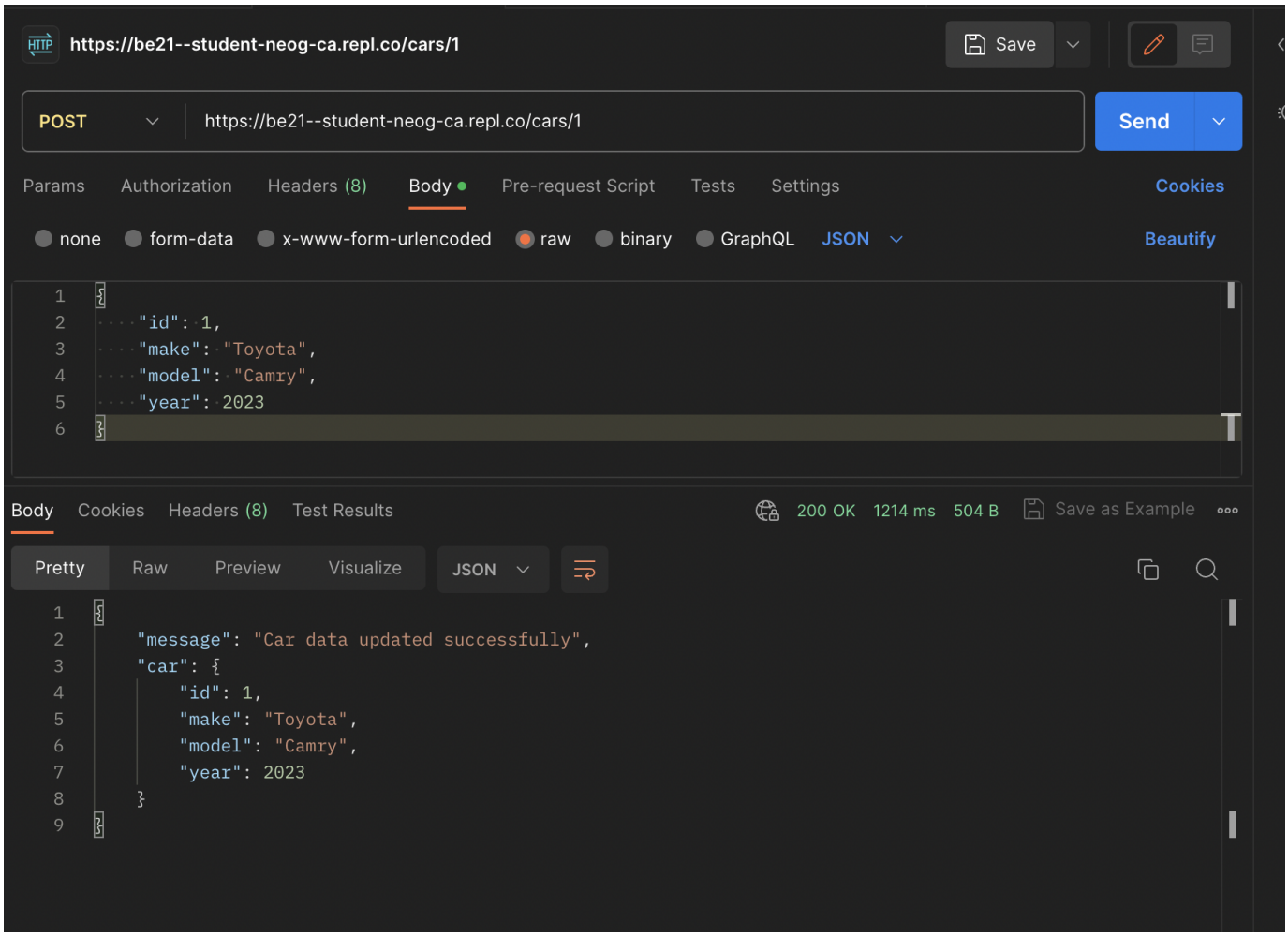
COPY

testing

Test Data Validation:

In Postman, send a POST request to `https://your-replit-app-url.glitch.me/cars` with incomplete data (missing make, model, or year) in the request body. Observe the error response.

- Solution



ex04: updating car data with POST requests (using car id)

In this exercise, we'll learn how to use POST requests to update car data based on the car's ID.

challenge

Create POST Route for Updating Car Data: Define a POST route at `/cars/:id` that receives updated car data in the request body and updates the corresponding car's information.

```
Design of APIs

/cars - GET will get all cars, POST will create a new car
/cars/:id - GET will get the id car, POST will update that id car
```

COPY

solution

```
app.post('/cars/:id', (req, res) => {
  const carId = parseInt(req.params.id)
  let isCarFound = false
  let updatedCar
  const updatedCarData = req.body

  cars = cars.map((car) => {
```

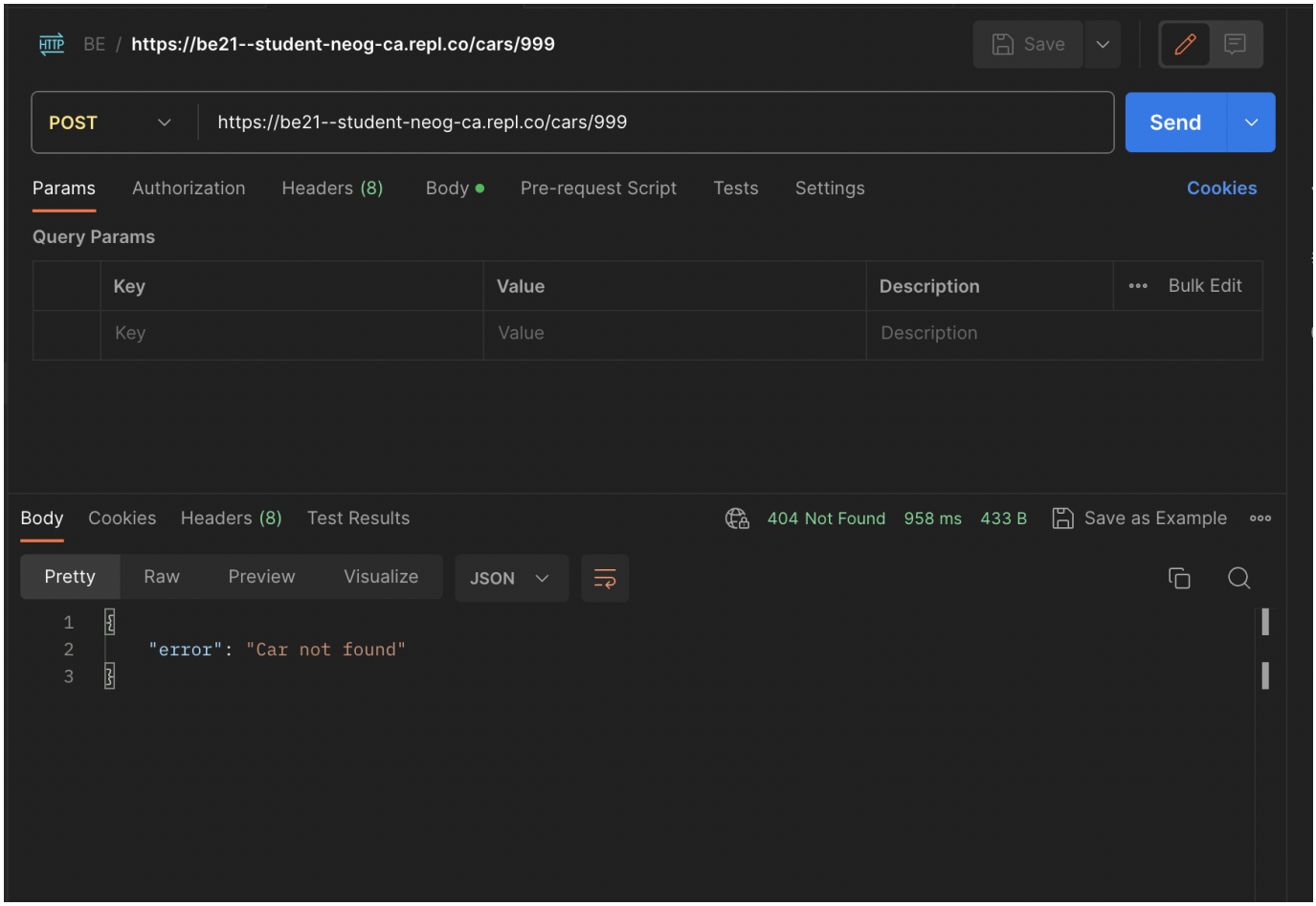
```
    if (car.id === carId) {
      isCarFound = true
      Object.assign(car, updatedCarData)
      updatedCar = car
    }
    return car
  })

  if (!isCarFound) {
    res.status(404).json({ error: 'Car not found' })
  } else {
    res.json({ message: 'Car data updated successfully', car: updatedCar })
  }
})
```

COPY

testing

Test Car Data Update: In Postman, send a POST request to `https://your-replit-app-url.glitch.me/cars/1` with updated car data in the request body. Observe the response and check if the car data has been updated.

- Solution



ex05: implementing error handling for car data updates

Now, let's enhance our car data update mechanism by incorporating error handling.

challenge

Update POST Route for Car Data Updates: Improve the `/cars/:id` route to include error handling for updating car data.

solution

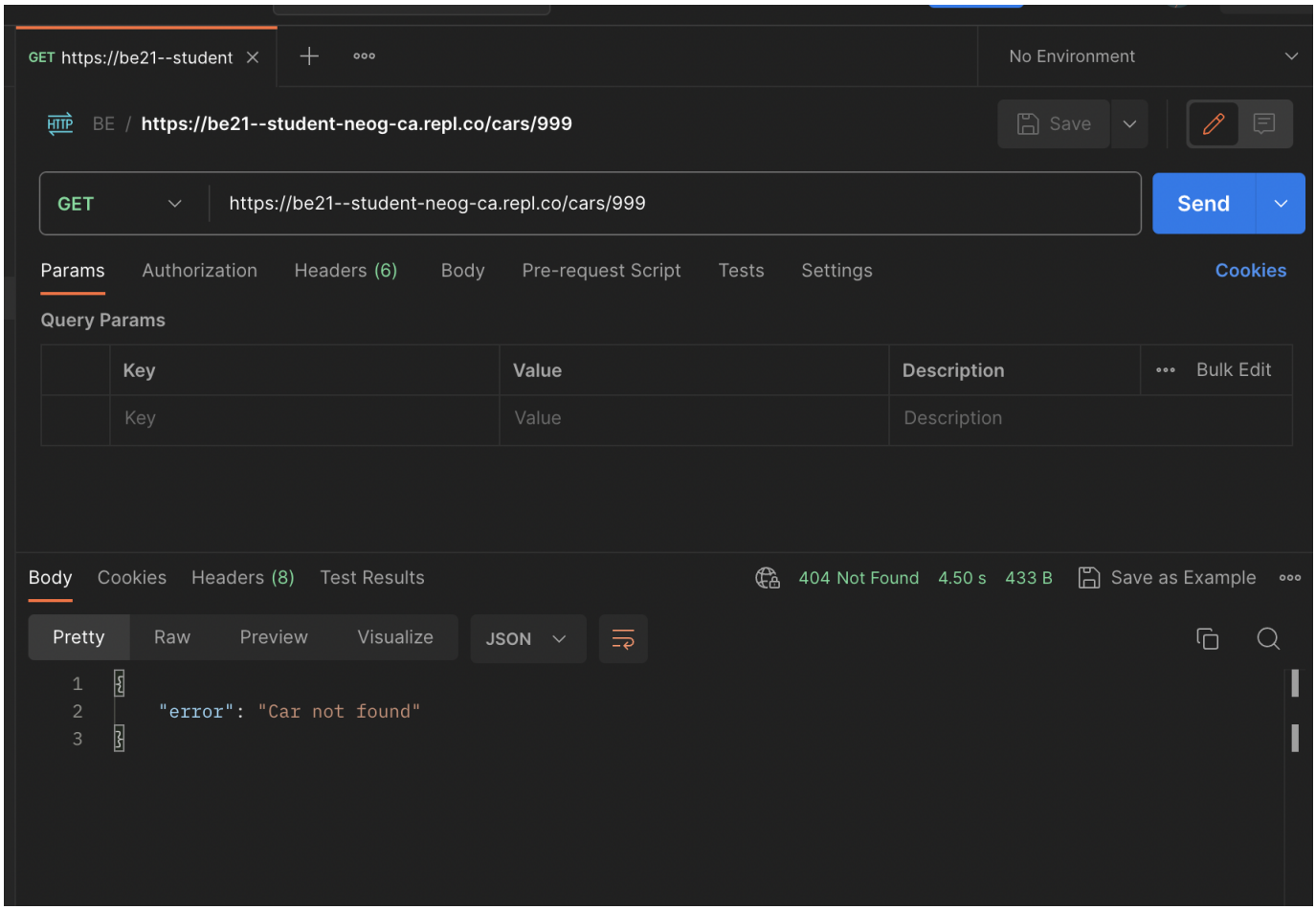
```
app.post('/cars/:id', (req, res) => {
  const carId = parseInt(req.params.id)
  const updatedCarData = req.body

  const carToUpdate = cars.find((car) => car.id === carId)

  if (!carToUpdate) {
    res.status(404).json({ error: 'Car not found' })
  } else {
    if (!updatedCarData.make || !updatedCarData.model || !updatedCarData.year) {
      res.status(400).json({ error: 'Make, model, and year are required' })
    } else {
      Object.assign(carToUpdate, updatedCarData)
      res.json({ message: 'Car data updated successfully', car: carToUpdate })
    }
  }
})
```

COPY

testing

Test Error Handling for Updates: In Postman, send a POST request to `https://your-replit-app-url.glitch.me/cars/999` (where 999 is a non-existent car ID) to observe the 404 error response. Then, send a POST request with incomplete updated car data to observe the 400 error response.

- Solution



final solution

https://replit.com/@tanaypratap/BE23CW