Backend 2.2_CW Exercises

variations of GET requests

ex01: creating a basic GET request to return JSON data

Let's start by creating a basic GET request that returns JSON data of cars from our inventory.

1. Define Cars Data:

   Create an array of car objects in your `index.js` file.

```
const cars = [
  { id: 1, make: 'Toyota', model: 'Camry', year: 2022 },
  { id: 2, make: 'Honda', model: 'Civic', year: 2021 },
  { id: 3, make: 'Ford', model: 'Mustang', year: 2022 },
  { id: 4, make: 'Chevrolet', model: 'Corvette', year: 2023 },
  { id: 5, make: 'Tesla', model: 'Model 3', year: 2021 },
  { id: 6, make: 'Nissan', model: 'Altima', year: 2022 },
  { id: 7, make: 'BMW', model: 'X5', year: 2023 },
  { id: 8, make: 'Mercedes-Benz', model: 'C-Class', year: 2021 },
  { id: 9, make: 'Audi', model: 'A4', year: 2022 },
  { id: 10, make: 'Lexus', model: 'RX', year: 2023 },
  { id: 11, make: 'Hyundai', model: 'Tucson', year: 2021 },
  { id: 12, make: 'Kia', model: 'Seltos', year: 2022 },
  { id: 13, make: 'Mazda', model: 'CX-5', year: 2023 },
  { id: 14, make: 'Subaru', model: 'Outback', year: 2021 },
  { id: 15, make: 'Volkswagen', model: 'Golf', year: 2022 },
]
```

COPY

2. Create GET Route for All Cars:

   Define a GET route at `/cars` that sends back the JSON data of all cars.

   Syntax:

```
app.get('<your-route>', (req, res) => {
  **res.json**(<data>); // variable, object, array
});
```

COPY

solution

```
app.get('/cars', (req, res) => {
  res.json(cars)
})
```

COPY

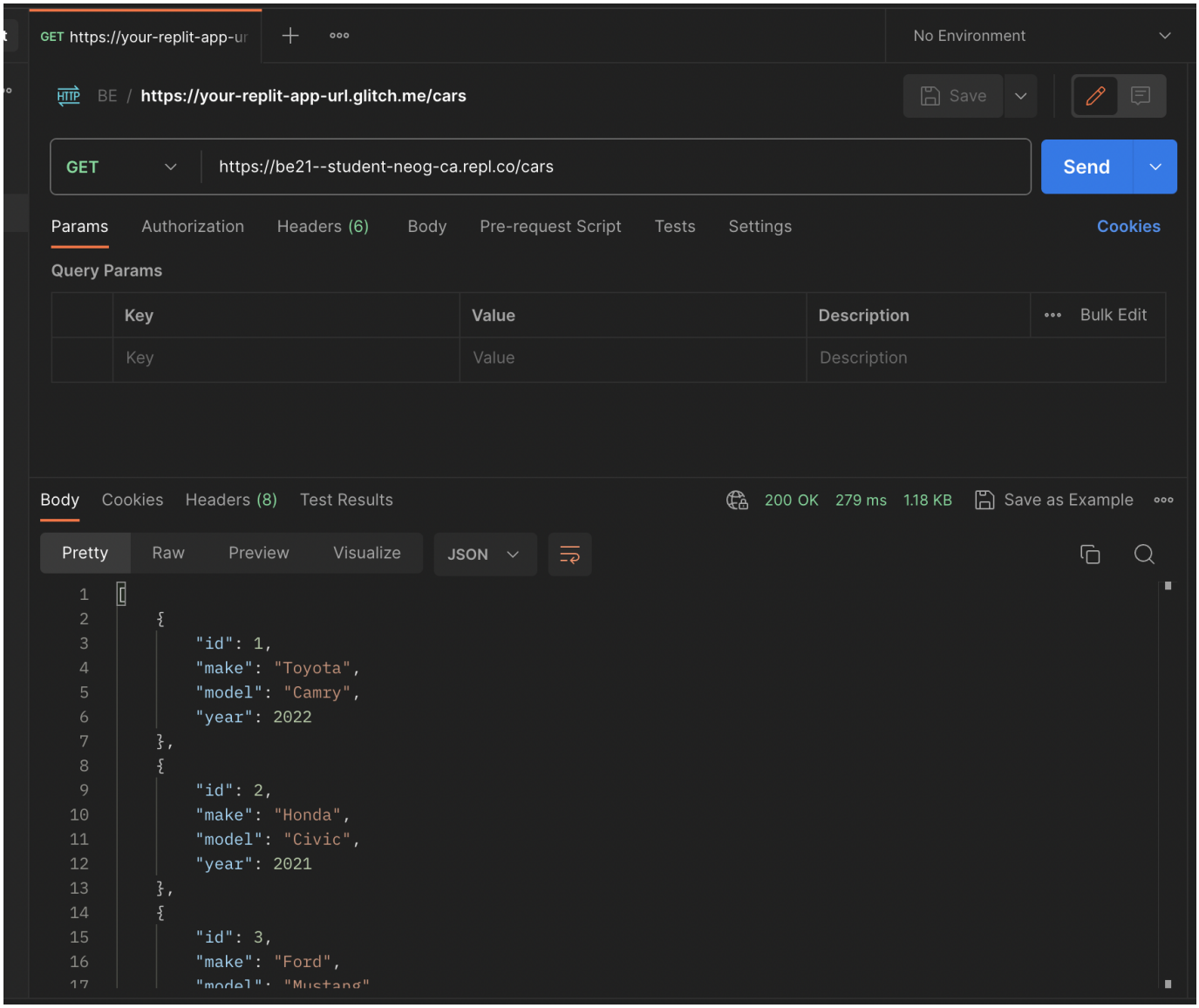
3. Testing

   Test in Postman:

   You need to restart your server.

   Use Postman to make a GET request to `https://your-replit-app-url.glitch.me/cars` and observe the JSON response.

solution



ex02: using query parameters in a GET request

In this exercise, we'll enhance our GET request by allowing users to filter cars based on query parameters.

challenge

Modify GET Route for Cars: Update the `/cars` route to handle query parameters.

Syntax:

```
app.get('/cars', (req, res) => {
  const { make, model } = req.query; // you can access the query data like this

        // your code here...
}
```

COPY

- solution

```
app.get('/cars', (req, res) => {
  const { make, model } = req.query
  let filteredCars = cars

  if (make) {
    filteredCars = filteredCars.filter(
      (car) => car.make.toLowerCase() === make.toLowerCase(),
    )
  }
  if (model) {
    filteredCars = filteredCars.filter(
      (car) => car.model.toLowerCase() === model.toLowerCase(),
    )
  }

  res.json(filteredCars)
})
```

COPY

testing

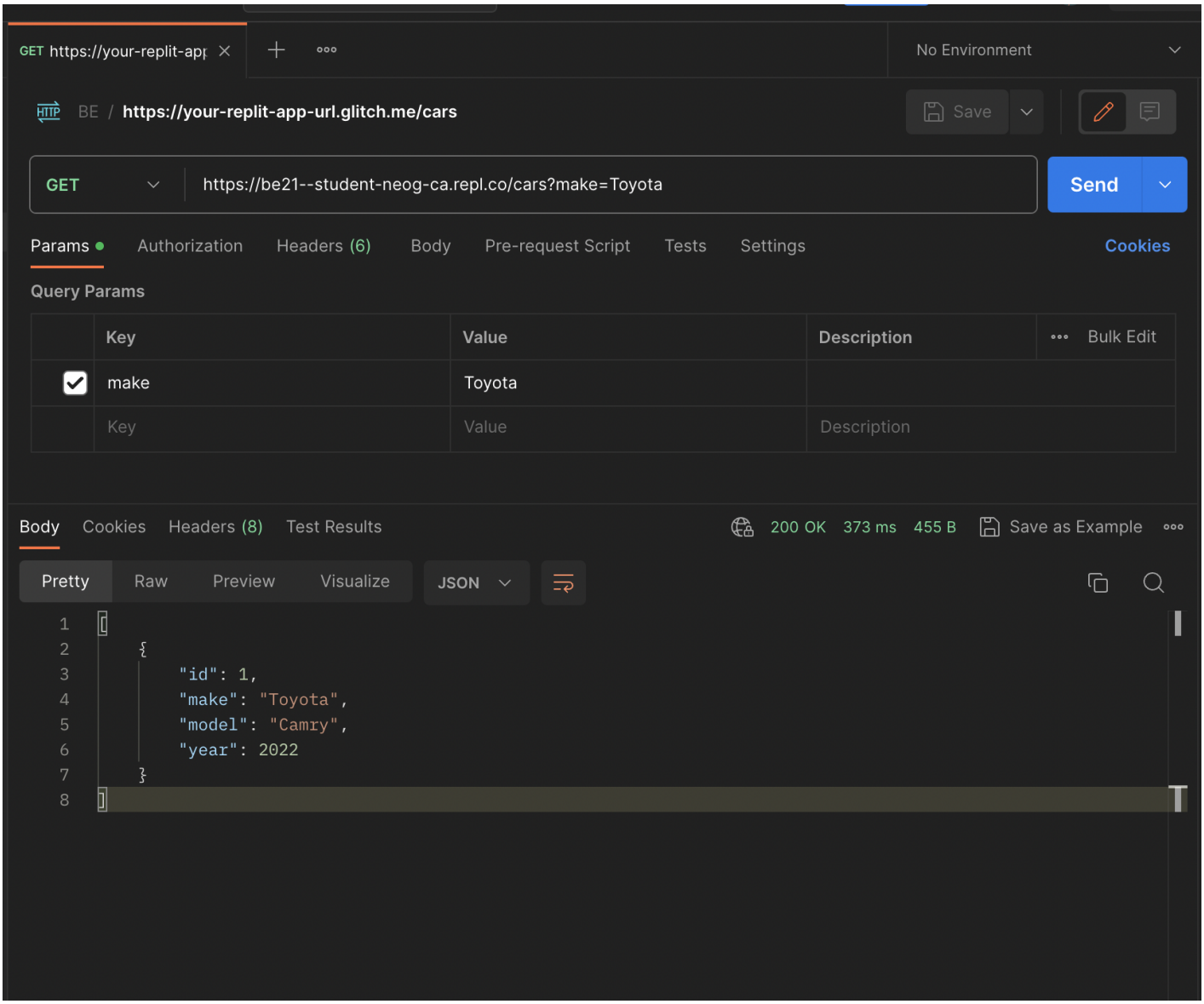
Test with Query Parameters: You need to restart your server.

In Postman, make GET requests

like `https://your-replit-app-url.glitch.me/cars?make=Toyota` or

`https://your-replit-app-url.glitch.me/cars?model=Civic` to see filtered results.

solution



ex03: using route parameters in a GET request

Let's now explore using route parameters to retrieve a specific car by its ID.

challenge

Create GET Route for Single Car: Define a GET route at `/cars/:id` to retrieve a car by its ID.

Syntax:

```
app.get('/cars/:id', (req, res) => {
        // You can retrieve the car id with the help of req.params
        const carId = parseInt(req.params.id)
  const { id } = req.params;

        // .. your code here
}
```

COPY

- solution

```
app.get('/cars/:id', (req, res) => {
  const carId = parseInt(req.params.id)
  const car = cars.find((car) => car.id === carId)

  if (car) {
    res.json(car)
  } else {
    res.status(404).json({ error: 'Car not found' })
  }
})
```

COPY

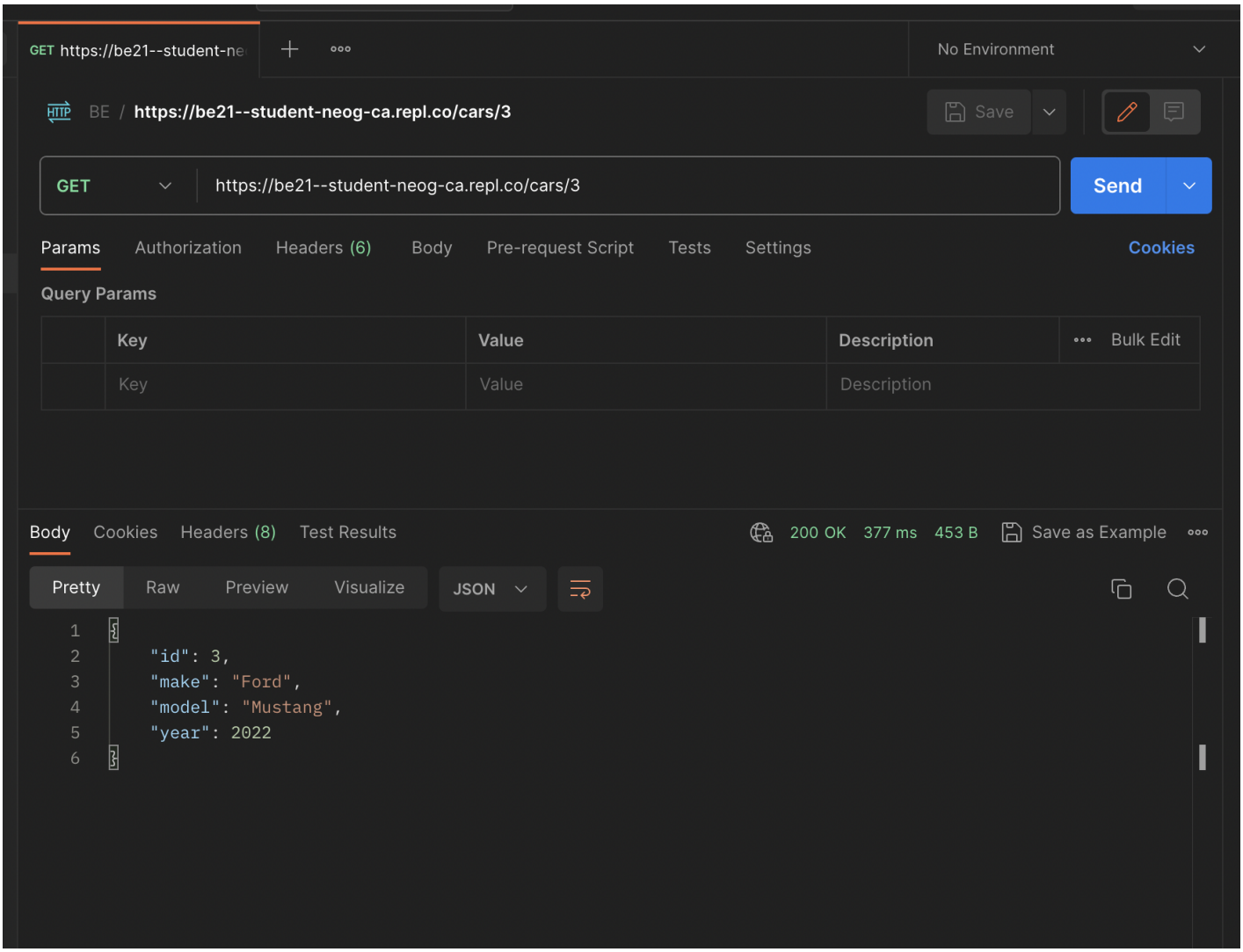
testing

Test with Route Parameters:

Make sure to restart your server.

In Postman, make GET requests like `https://your-replit-app-url.glitch.me/cars/3` to retrieve a car by ID.

solution



ex04: using multiple route parameters in a GET request

Now, let's extend our knowledge by using multiple route parameters to retrieve cars based on make and model.

challenge

Create GET Route with Multiple Parameters:

Define a GET route at `/cars/:make/:model` to retrieve cars based on both make and model.

solution

```
app.get('/cars/:make/:model', (req, res) => {
  const make = req.params.make.toLowerCase()
  const model = req.params.model.toLowerCase()
  const filteredCars = cars.filter(
    (car) =>
      car.make.toLowerCase() === make && car.model.toLowerCase() === model,
```

```
  )

  res.json(filteredCars)
})
```

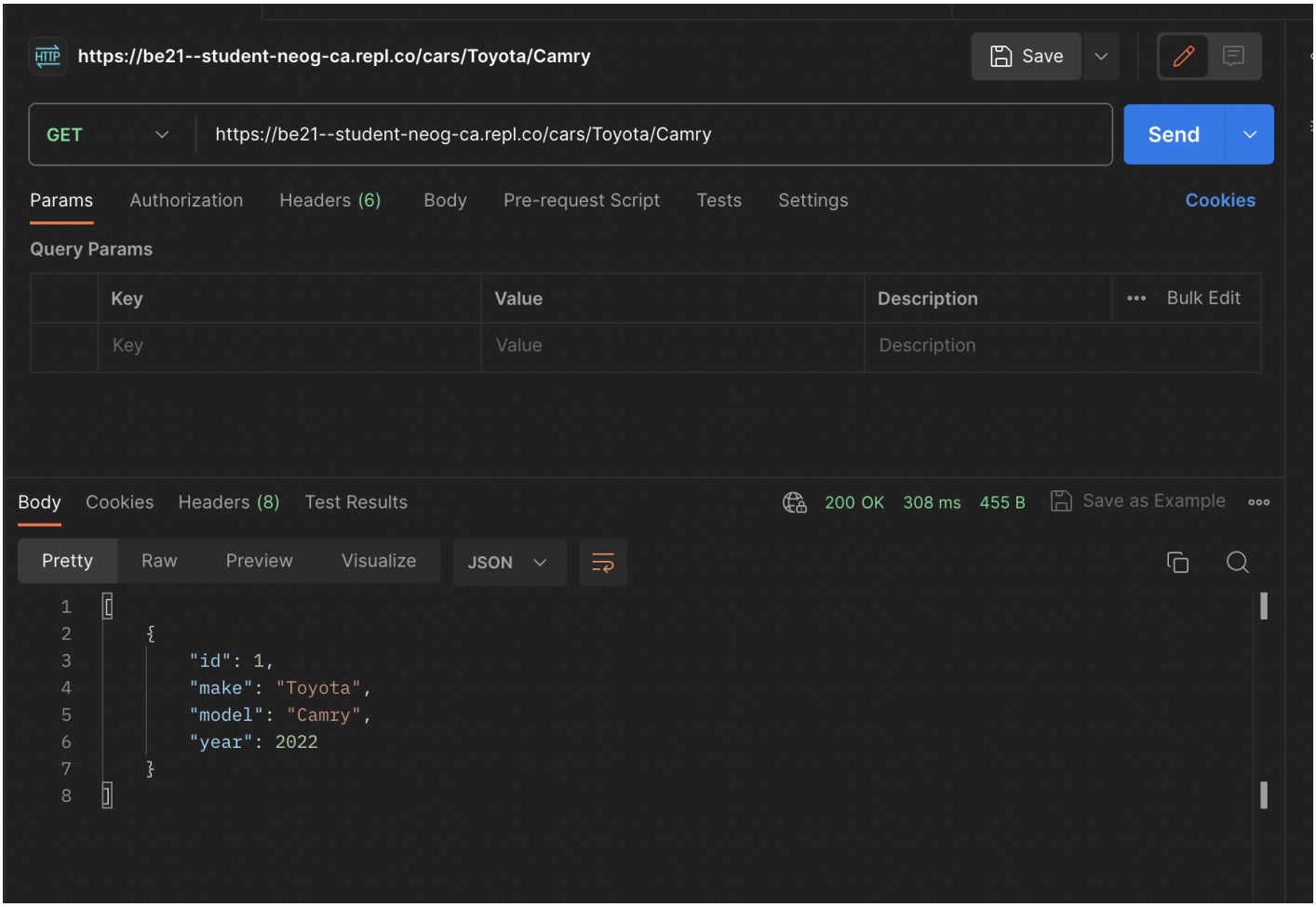
COPY

testing

Test with Multiple Route Parameters:

In Postman, make GET requests like `https://your-replit-app-url.glitch.me/cars/Toyota/Camry` to retrieve cars by make and model.

solution



ex05: using headers in a GET request

In this final exercise, let's explore how to use headers to request specific data.

challenge

Create GET Route with Headers:

Define a GET route at `/cars/featured` that returns featured cars (which have year 2022 and above) based on a custom header.

📌 Make sure you place this request before `/cars/:id` route.

```
/cars --> 1
/cars/:id ---> 2 dynamic
/cars/featured --> 3 static // wrong and that's why you will not reach here

yourserver.com/cars // 1

yourserver.com/cars/1 // 2

yourserver.com/cars/tanay // 2
yourserver.com/cars/featured

Place our routes from specific to generic
or, static routes before dynamic ones

/cars/all before 1, after 1, after 2, after 3
```

COPY

solution

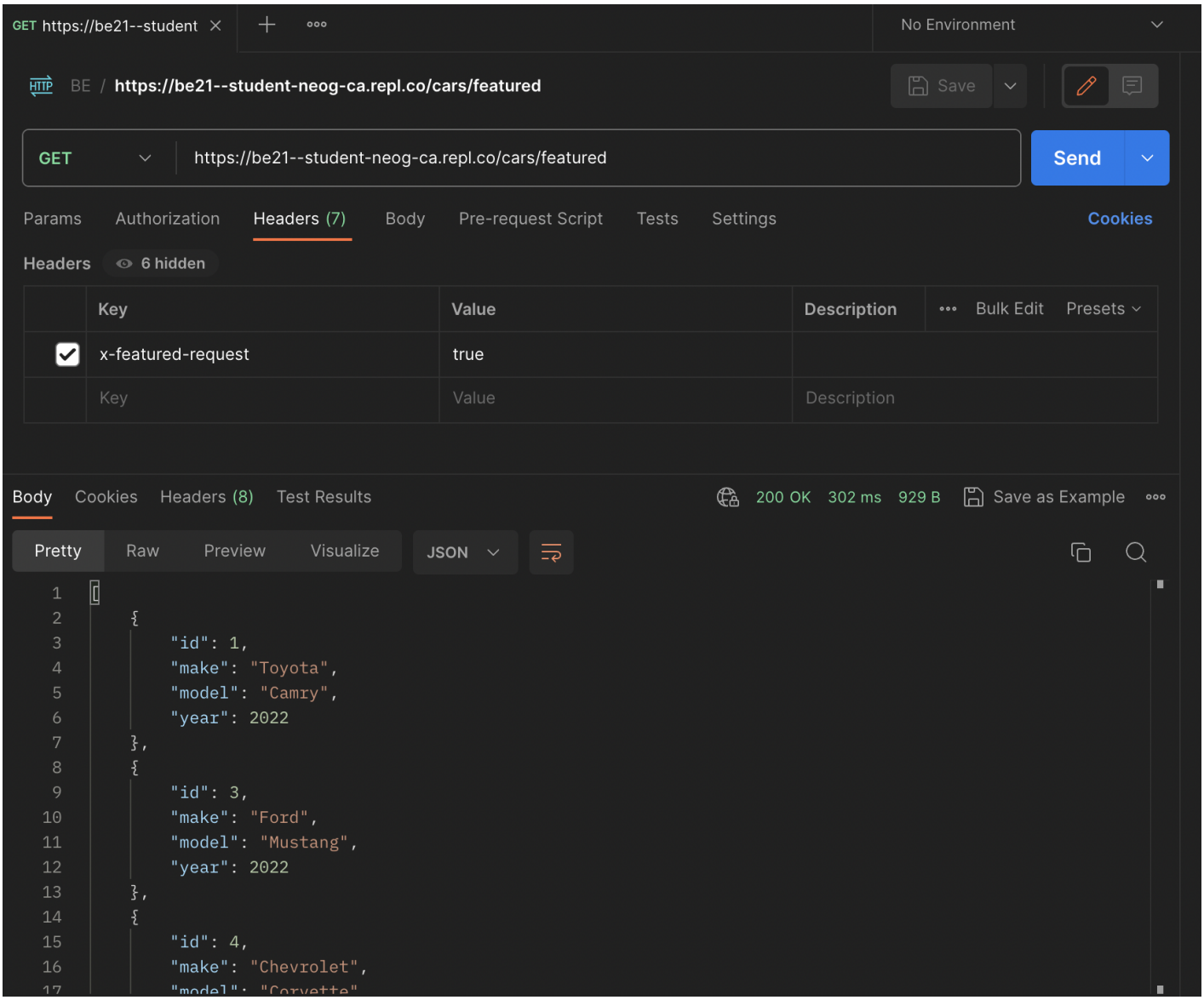
```
app.get('/cars/featured', (req, res) => {
  const isFeaturedRequest = req.header('x-featured-request')

  if (isFeaturedRequest === 'true') {
    const featuredCars = cars.filter((car) => car.year >= 2022)
    res.json(featuredCars)
  } else {
    res.status(400).json({ error: 'Invalid request' })
  }
})
```

COPY

testing

Test with Headers: In Postman, make a GET request to `https://your-replit-app-url.glitch.me/cars/featured` and set the header `x-featured-request` to `true` to see featured cars.

solution



ex06: implementing 404 error handling with query parameters

In this exercise, we'll enhance our knowledge by implementing error handling when a car with specific query parameters is not found.

challenge

Modify GET Route for Cars: Update the `/cars` route to handle query parameters.

solution

```
app.get('/cars', (req, res) => {
  const { make, model } = req.query
  let filteredCars = cars

  if (make) {
    filteredCars = filteredCars.filter(
```

```
      (car) => car.make.toLowerCase() === make.toLowerCase(),
    )
  }
  if (model) {
    filteredCars = filteredCars.filter(
      (car) => car.model.toLowerCase() === model.toLowerCase(),
    )
  }

  if (filteredCars.length === 0) {
    res.status(404).json({ error: 'No cars matching the query parameters' })
  } else {
    res.json(filteredCars)
  }
})
```

COPY

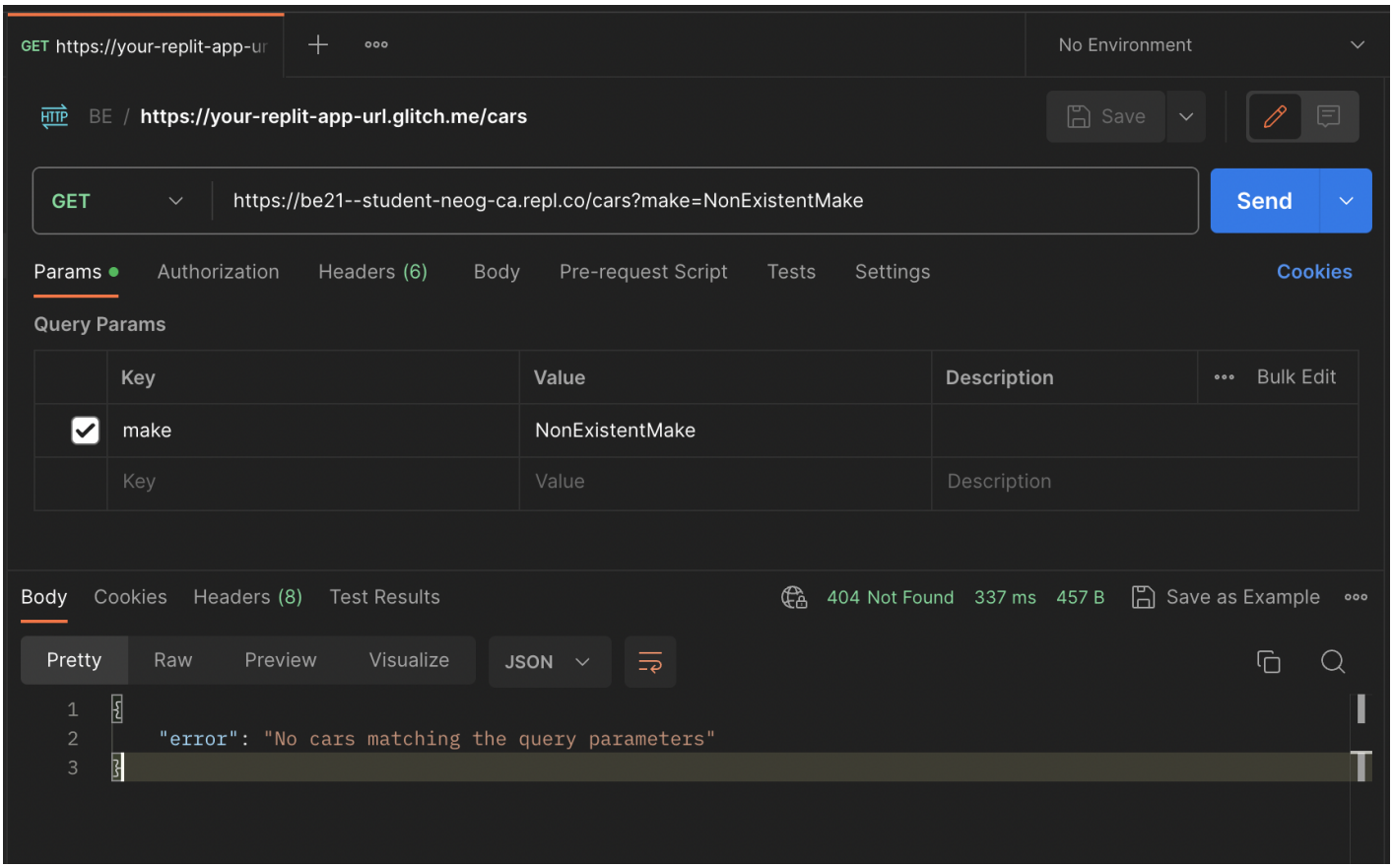
testing

Test 404 Error Handling:

In Postman, make a GET request like `https://your-replit-app-url.glitch.me/cars?make=Ford` to see cars matching the query parameters.

Then, try querying for a non-existent make like `https://your-replit-app-url.glitch.me/cars?make=NonExistentMake` to observe the 404 error response.

solution



ex07: implementing 404 error handling with route parameters

Now, let's extend our error handling knowledge to route parameters.

challenge

Create GET Route for Single Car: Define a GET route at `/cars/:id` to retrieve a car by its ID.

solution

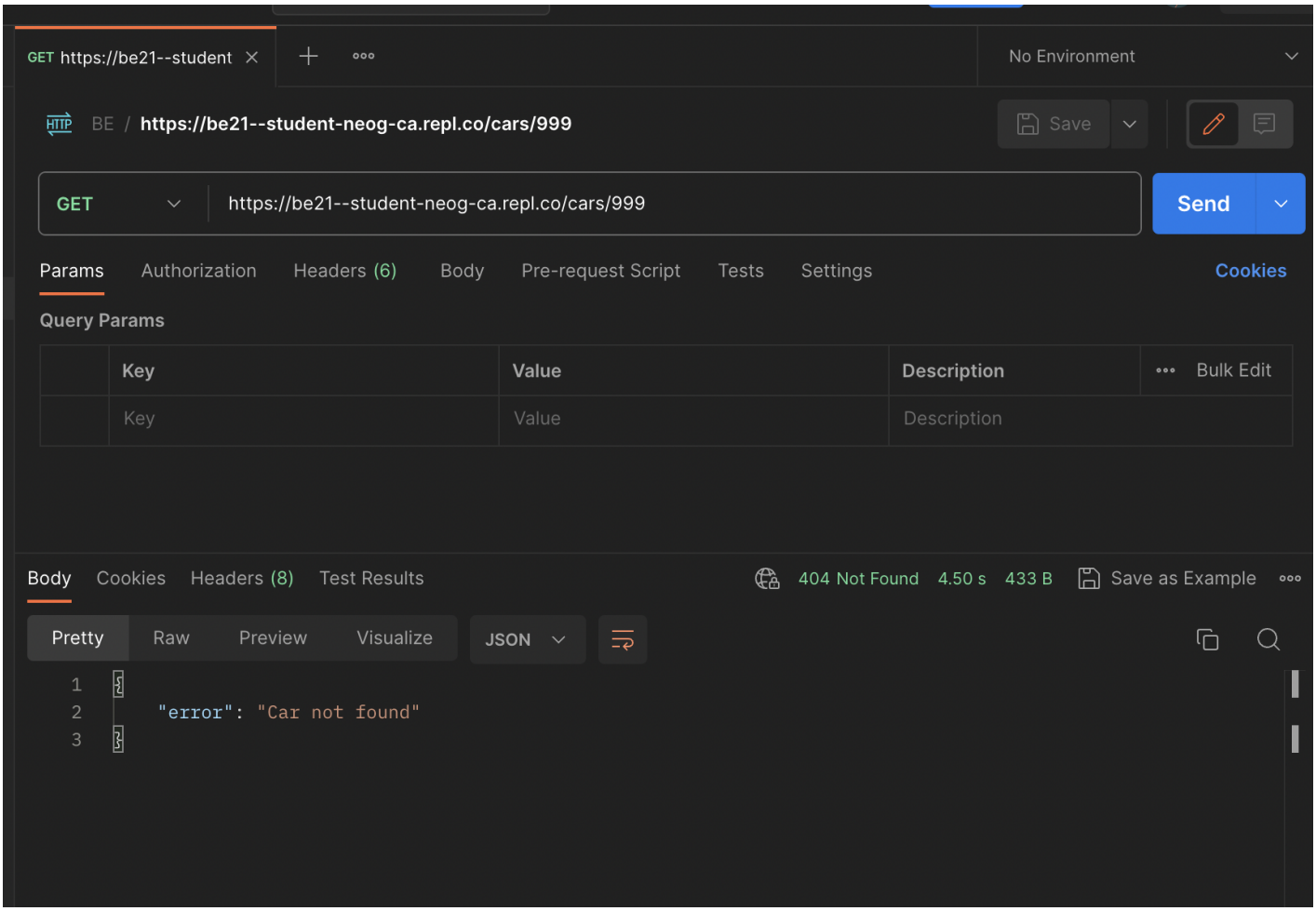
```
app.get('/cars/:id', (req, res) => {
  const carId = parseInt(req.params.id)
  const car = cars.find((car) => car.id === carId)

  if (car) {
    res.json(car)
  } else {
    res.status(404).json({ error: 'Car not found' })
  }
})
```

COPY

testing

Test 404 Error Handling: In Postman, make GET requests like `https://your-replit-app-url.glitch.me/cars/1` to retrieve a car by ID. Then, try querying for a non-existent ID like `https://your-replit-app-url.glitch.me/cars/999` to observe the 404 error response.

solution



final solution

https://replit.com/@tanaypratap/BE22CW