MC

Ramin Bahadoorifar

Creating a PostgresDB

I created my PostgresDB with a `docker-compose.yaml` file, which can be started by running `docker compose up -d`

```
version: '3.8'
services:
  postgres:
    image: postgres:14.1-alpine
    restart: always
    environment:
      - POSTGRES_USER=postgres
      - POSTGRES_PASSWORD=password
    ports:
      - '5432:5432'
    volumes:
      - postgres:/var/lib/postgresql/data
volumes:
  postgres:
    driver: local
```

Semaphore Tutorial

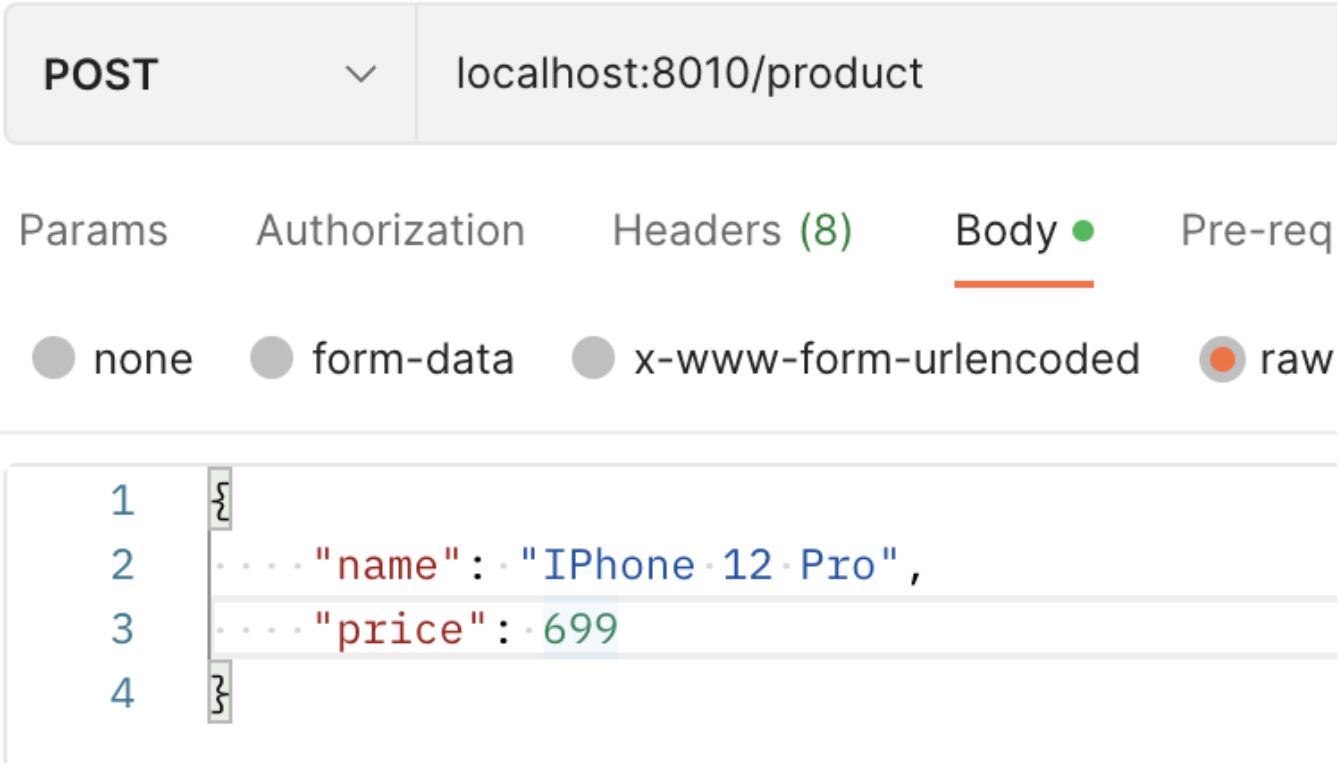
I followed the semaphore tutorial and tested it at the end by executing the tests.

```
ramin@Ramins-MacBook-Pro fh-mc-cicd-go % go test -v
=== RUN   TestEmptyTable
--- PASS: TestEmptyTable (0.01s)
=== RUN   TestGetNonExistentProduct
--- PASS: TestGetNonExistentProduct (0.01s)
=== RUN   TestCreateProduct
--- PASS: TestCreateProduct (0.01s)
=== RUN   TestGetProduct
--- PASS: TestGetProduct (0.01s)
=== RUN   TestUpdateProduct
--- PASS: TestUpdateProduct (0.01s)
=== RUN   TestDeleteProduct
--- PASS: TestDeleteProduct (0.01s)
PASS
ok      github.com/rambah/fh-mc-cicd-go 0.204s
```

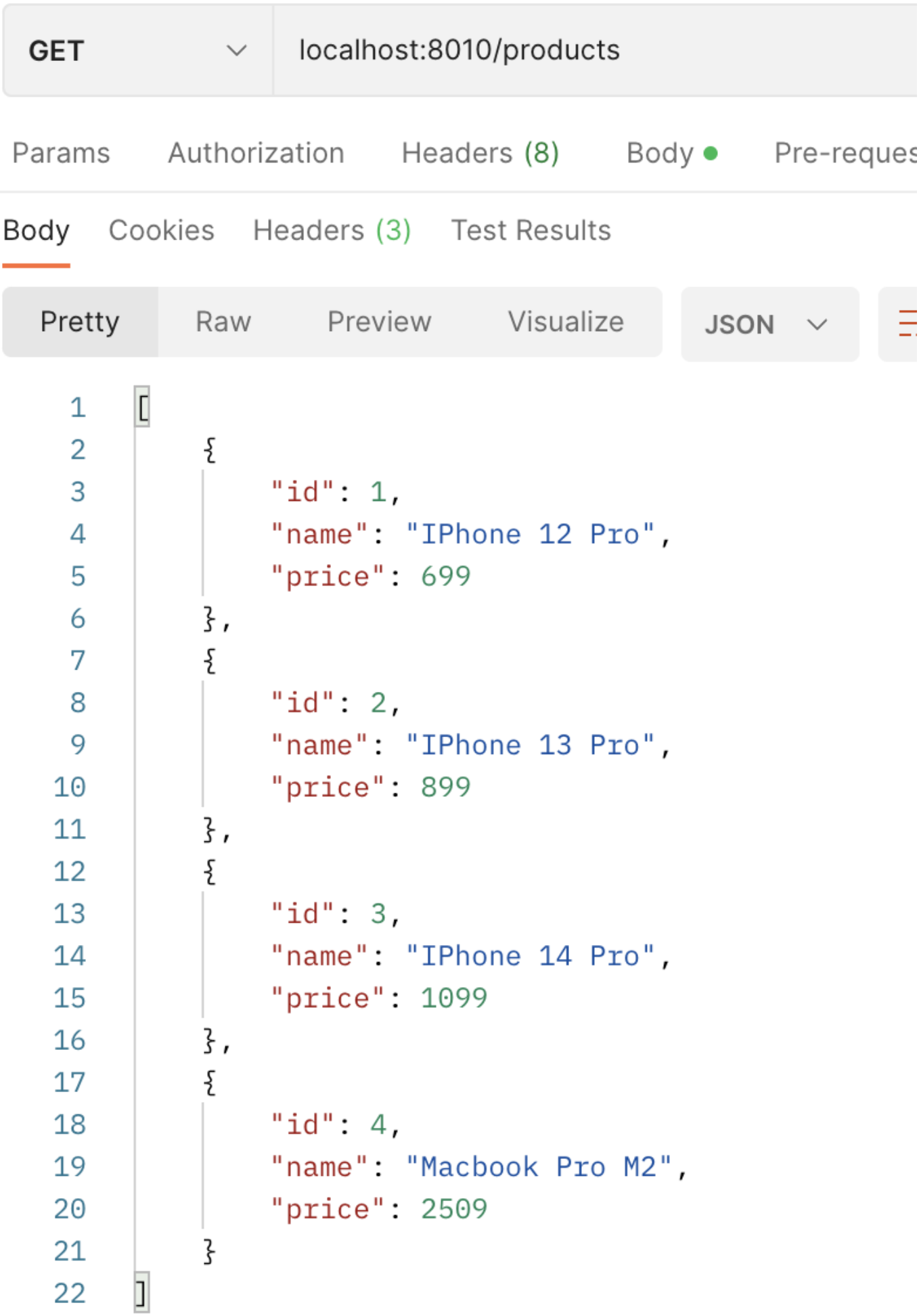
After that I started the server with

```
go run .
```

and run some POST Methods with Postman:



After that I executed a GET Method to get all products:


GET
localhost:8010/products
Params
Authorization
Headers (8)
Body
Pre-reques
Body
Cookies
Headers (3)
Test Results
Pretty
Raw
Preview
Visualize
JSON
[
{
"id": 1,
"name": "IPhone 12 Pro",
"price": 699
},
{
"id": 2,
"name": "IPhone 13 Pro",
"price": 899
},
{
"id": 3,
"name": "IPhone 14 Pro",
"price": 1099
},
{
"id": 4,
"name": "Macbook Pro M2",
"price": 2509
}
]

Added Features

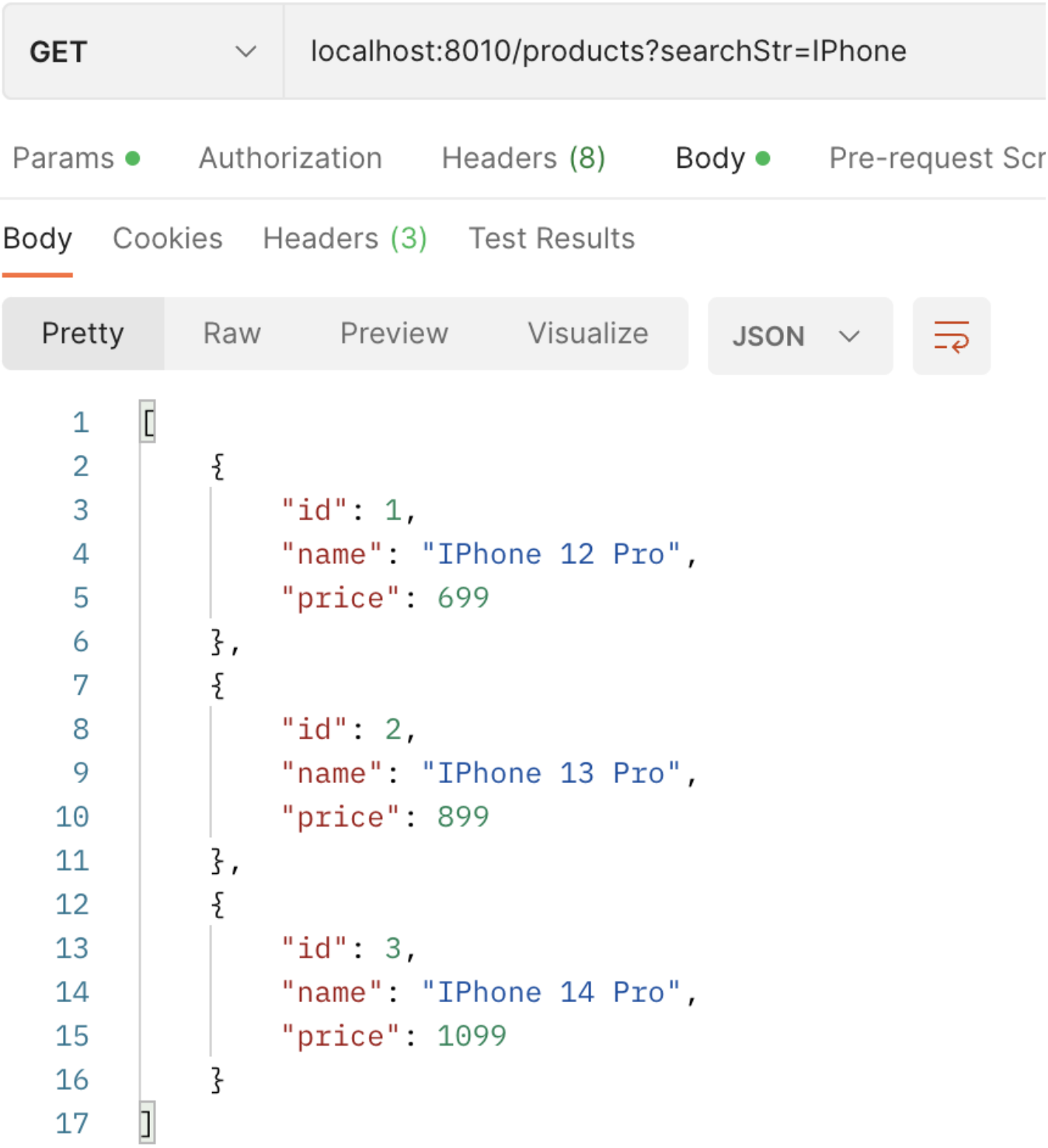
Search by String

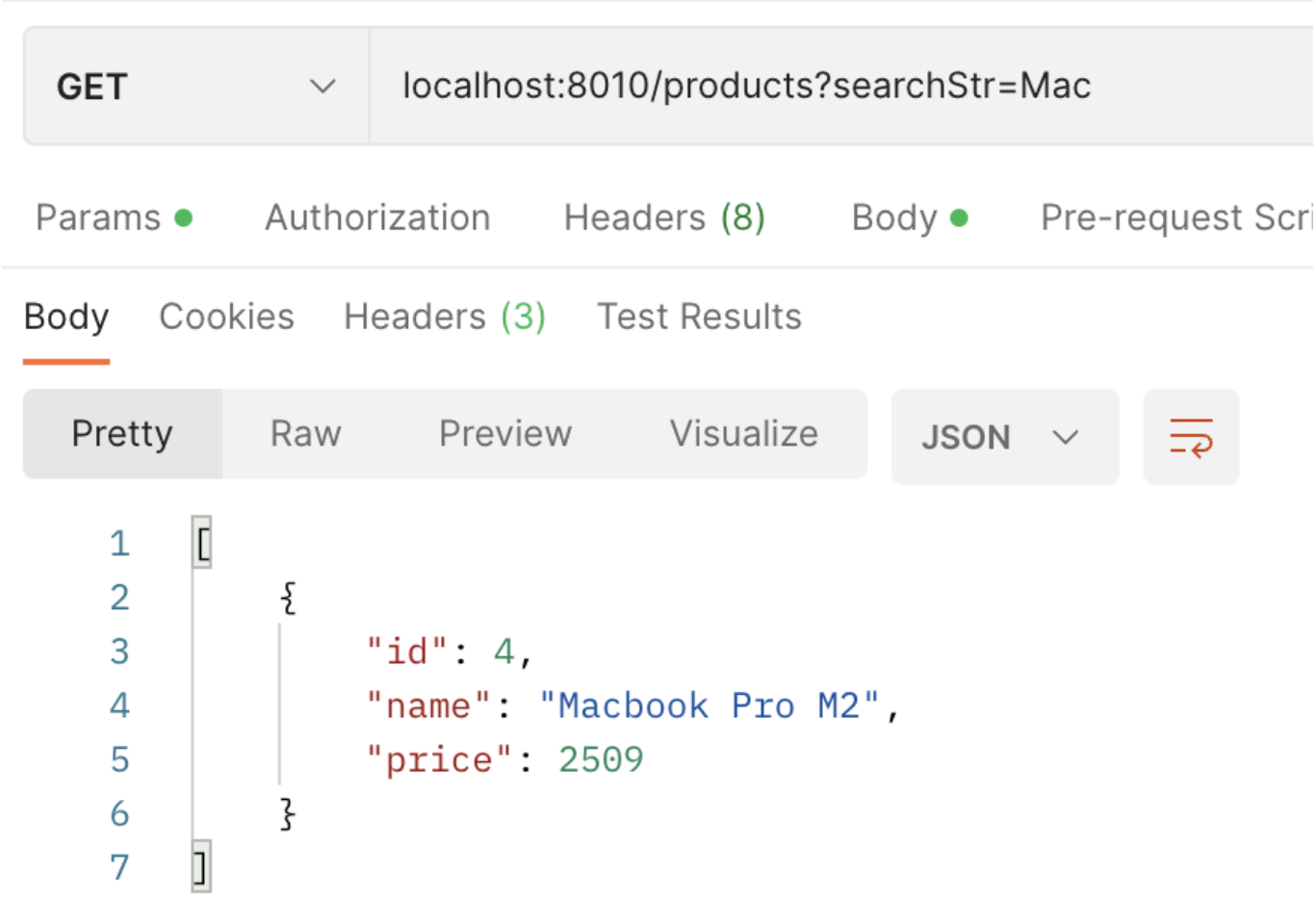
I modified the `getProducts(...)` function with a search feature.

```
// app.go
func (a *App) getProducts(w http.ResponseWriter, r *http.Request) {
 count, _ := strconv.Atoi(r.FormValue("count"))
 start, _ := strconv.Atoi(r.FormValue("start"))
 searchStr := r.FormValue("searchStr") // <--- I added this line

 // [...]

 products, err := getProducts(a.DB, start, count, searchStr)
}
```

```
// model.go
func getProducts(db *sql.DB, start, count int, searchStr string)
([]product, error) {
 var rows *sql.Rows
 var err error
 if searchStr == "" {
  rows, err = db.Query(
   "SELECT id, name,  price FROM products LIMIT $1 OFFSET $2",
   count, start)
 } else {
  rows, err = db.Query(
   "SELECT id, name, price FROM products WHERE LOWER(name) LIKE
LOWER('%'||$1||'%') LIMIT $2 OFFSET $3", searchStr, count, start,
  )
 }
 [...]
```

Test with Postman:


GET
localhost:8010/products?searchStr=IPhone
Params
Authorization
Headers (8)
Body
Pre-request Scr
Body
Cookies
Headers (3)
Test Results
Pretty
Raw
Preview
Visualize
JSON
[
{
"id": 1,
"name": "IPhone 12 Pro",
"price": 699
},
{
"id": 2,
"name": "IPhone 13 Pro",
"price": 899
},
{
"id": 3,
"name": "IPhone 14 Pro",
"price": 1099
}
]



Reset Database

An Endpoint to reset the database.

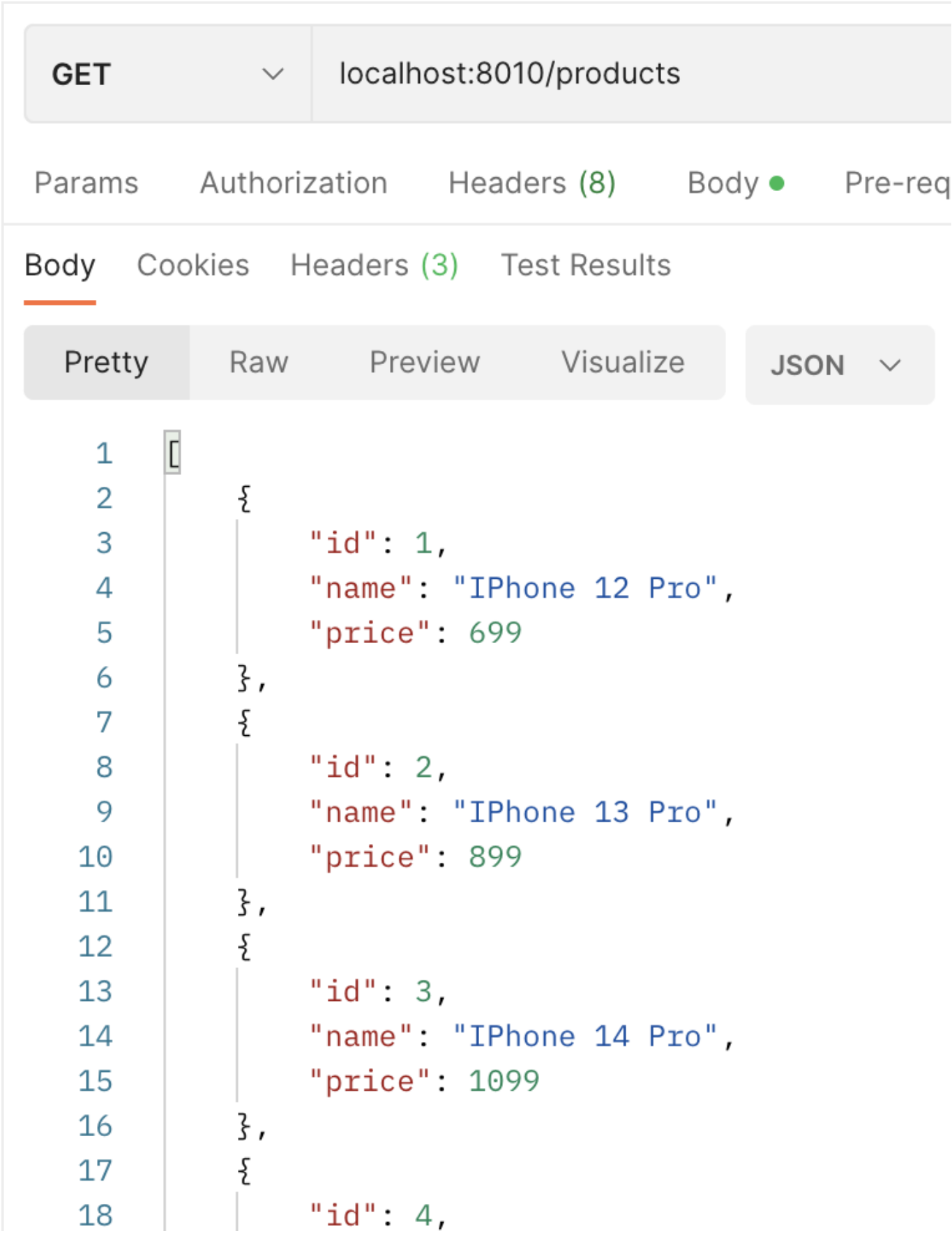
```
// app.go
func (a *App) resetDatabase(w http.ResponseWriter, r *http.Request) {
 if err := resetDatabase(a.DB); err != nil {
  respondWithError(w, http.StatusInternalServerError, err.Error())
  return
 }

 respondWithJSON(w, http.StatusOK, map[string]string{"result": "success"})
}
```

```
// model.go
func resetDatabase(db *sql.DB) error {
 _, err := db.Exec("DELETE FROM products")
 if err != nil {
  return err
 }
 _, err = db.Exec("ALTER SEQUENCE products_id_seq RESTART WITH 1")
```

```
  return err
}
```

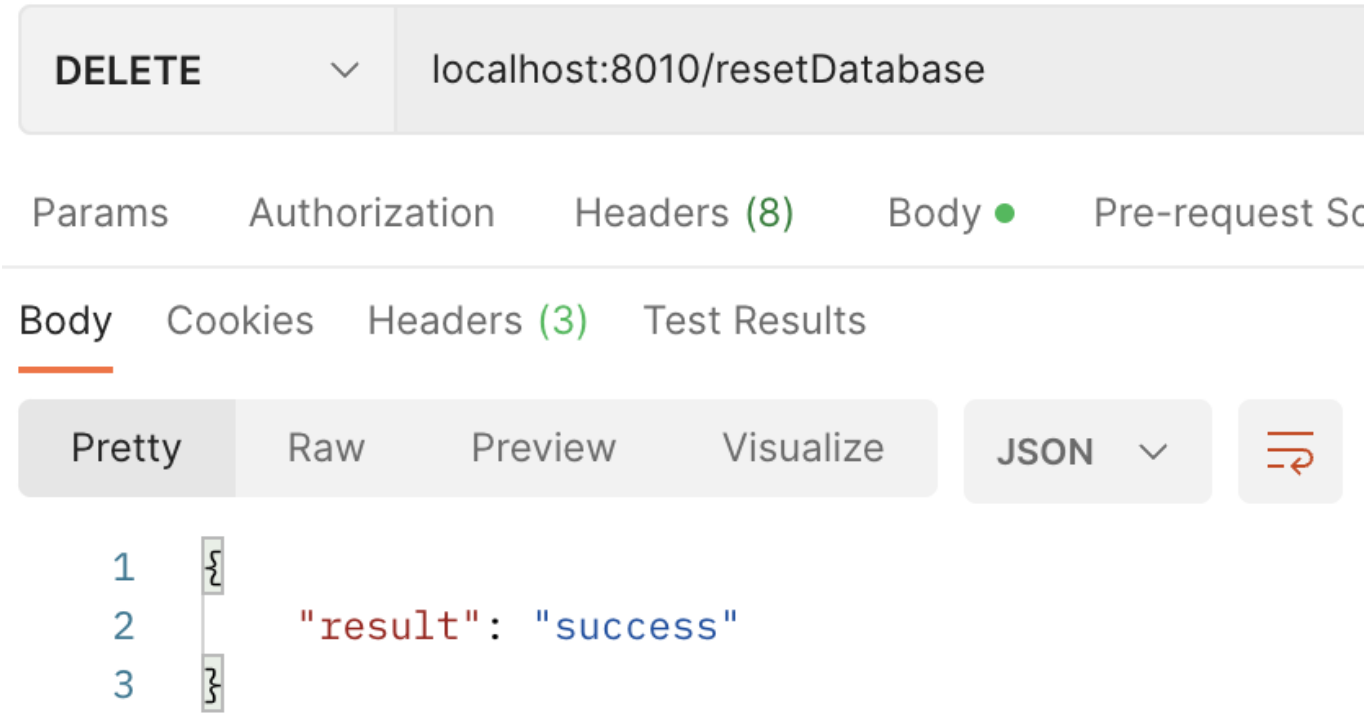
Test with Postman:

Database with filled products.

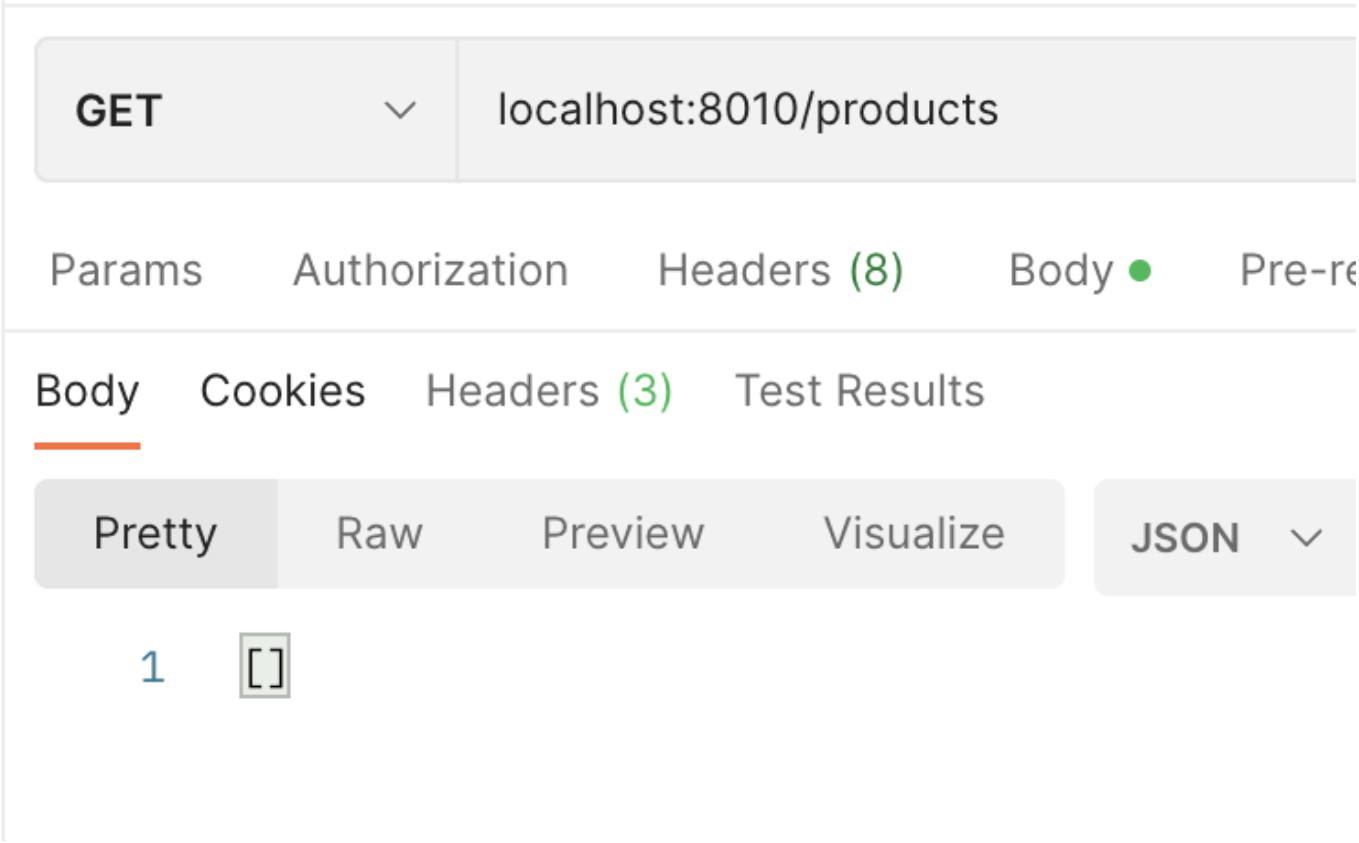


```
        "name": "Macbook Pro M2",
        "price": 2509
    }
]
```

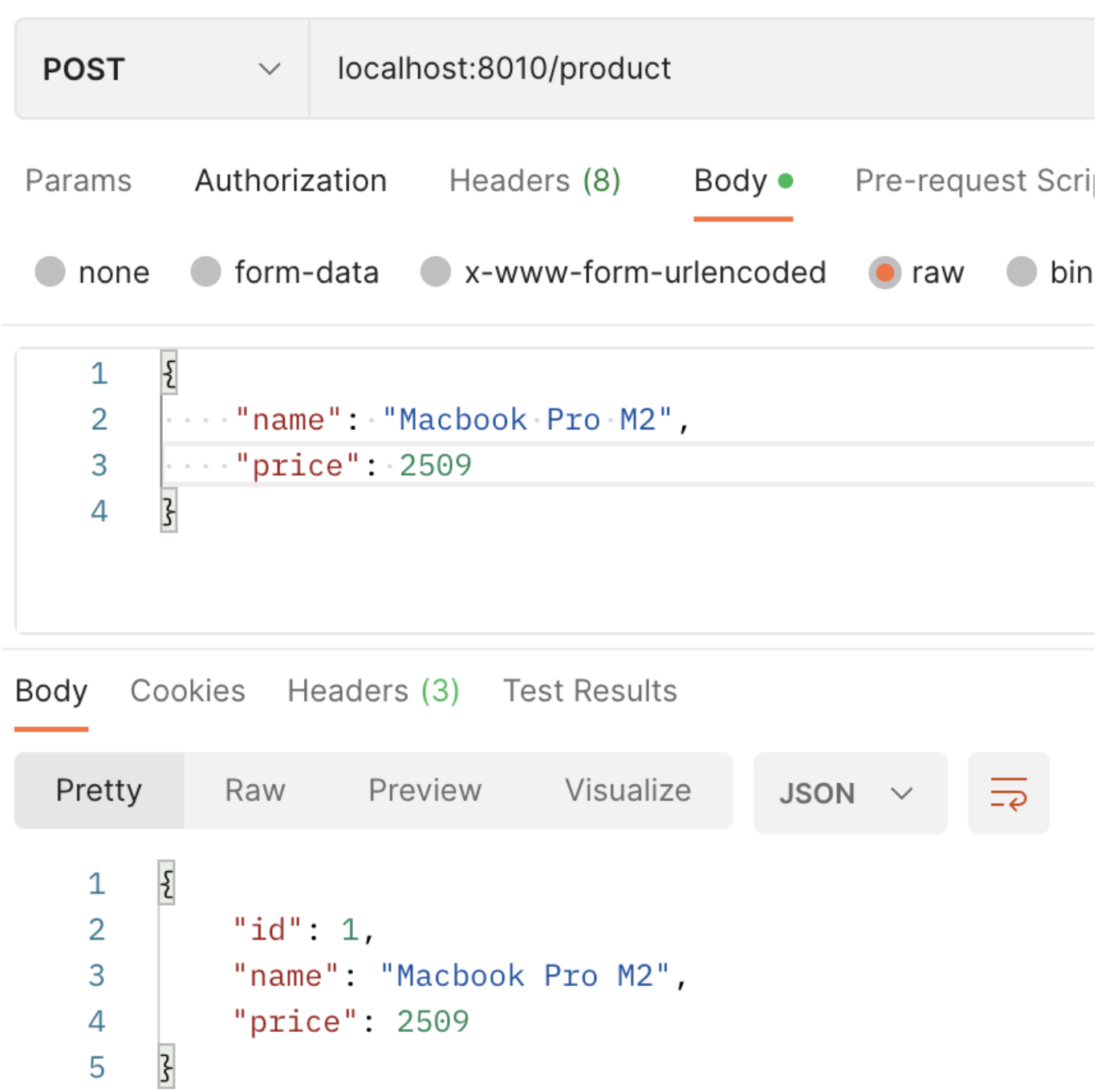
Reset the Database.



Test if all products have been deleted.



Create a Product, which should begin with the ID 1 again.


POST
localhost:8010/product
Params
Authorization
Headers (8)
Body
Pre-request Scri
none
form-data
x-www-form-urlencoded
raw
bin
1 {
2 "name": "Macbook Pro M2",
3 "price": 2509
4 }
Body
Cookies
Headers (3)
Test Results
Pretty
Raw
Preview
Visualize
JSON
1 {
2 "id": 1,
3 "name": "Macbook Pro M2",
4 "price": 2509
5 }