Department of Statistics & Computer Science
University of Kelaniya
ACADEMIC YEAR – 2021/2022
COSC 32133
Full-Stack Software Development
Practical Tutorial 03

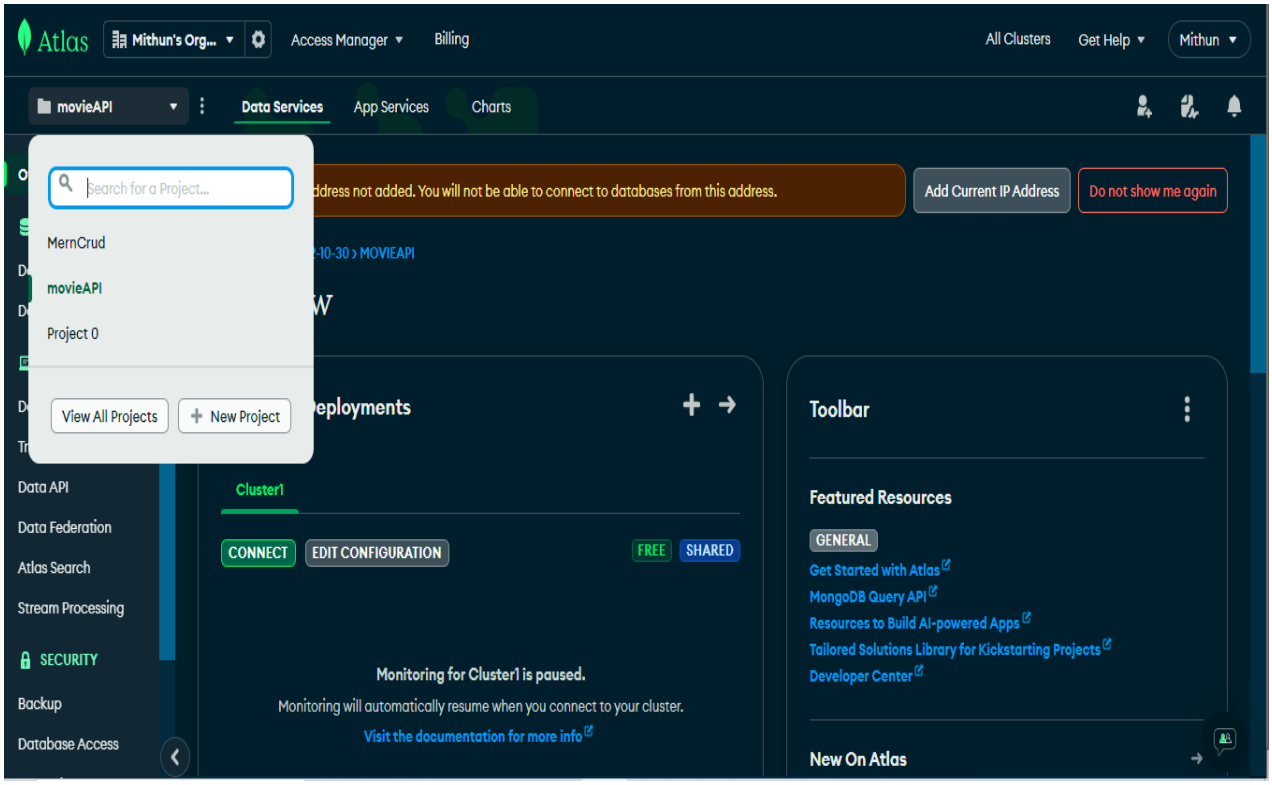
Step 01: Deploy a project in the Mongodb Atlas environment.

1. Go to the MongoDB Atlas website at https://www.mongodb.com/cloud/atlas .

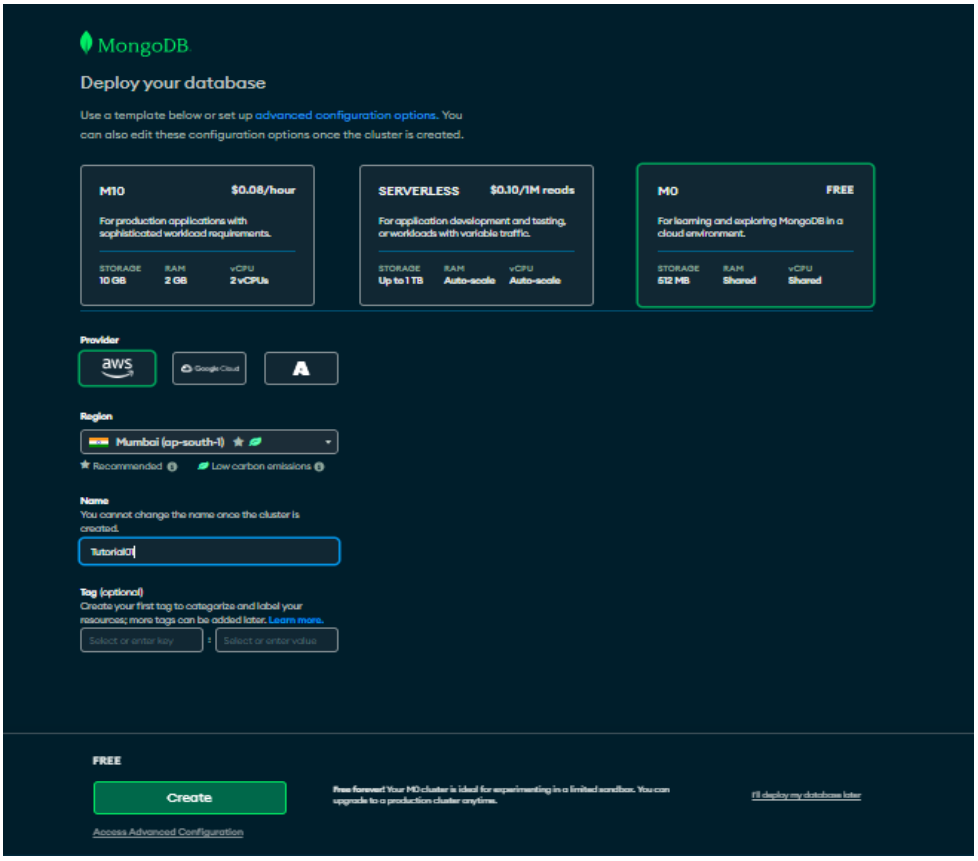
2. Click "Start Free" or "Try Free".

3. Complete the sign-up form with your details, including your name, email address, and password or sign up with your Gmail address.

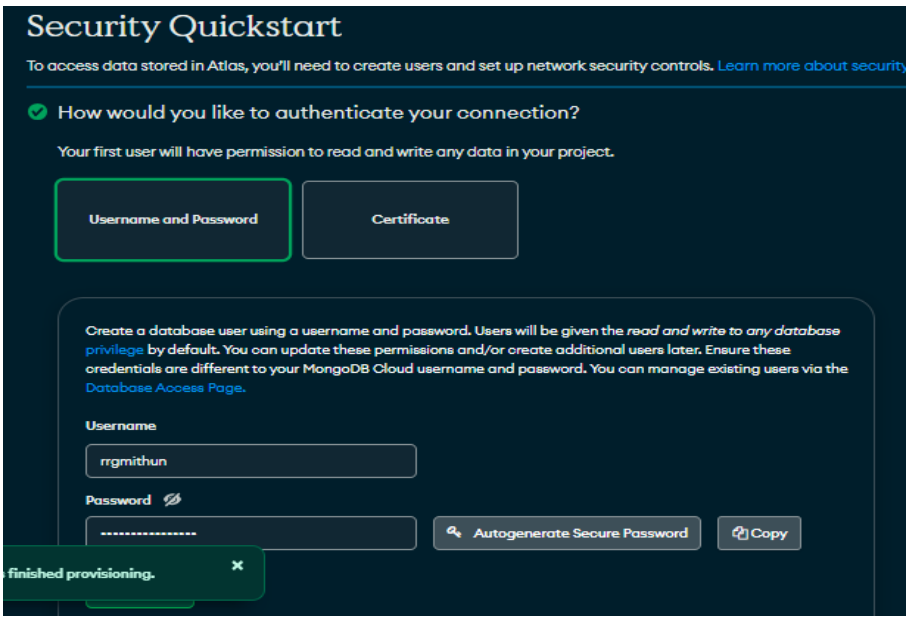
4.Create a new project by clicking the 'New Project' button and give it a valid name.

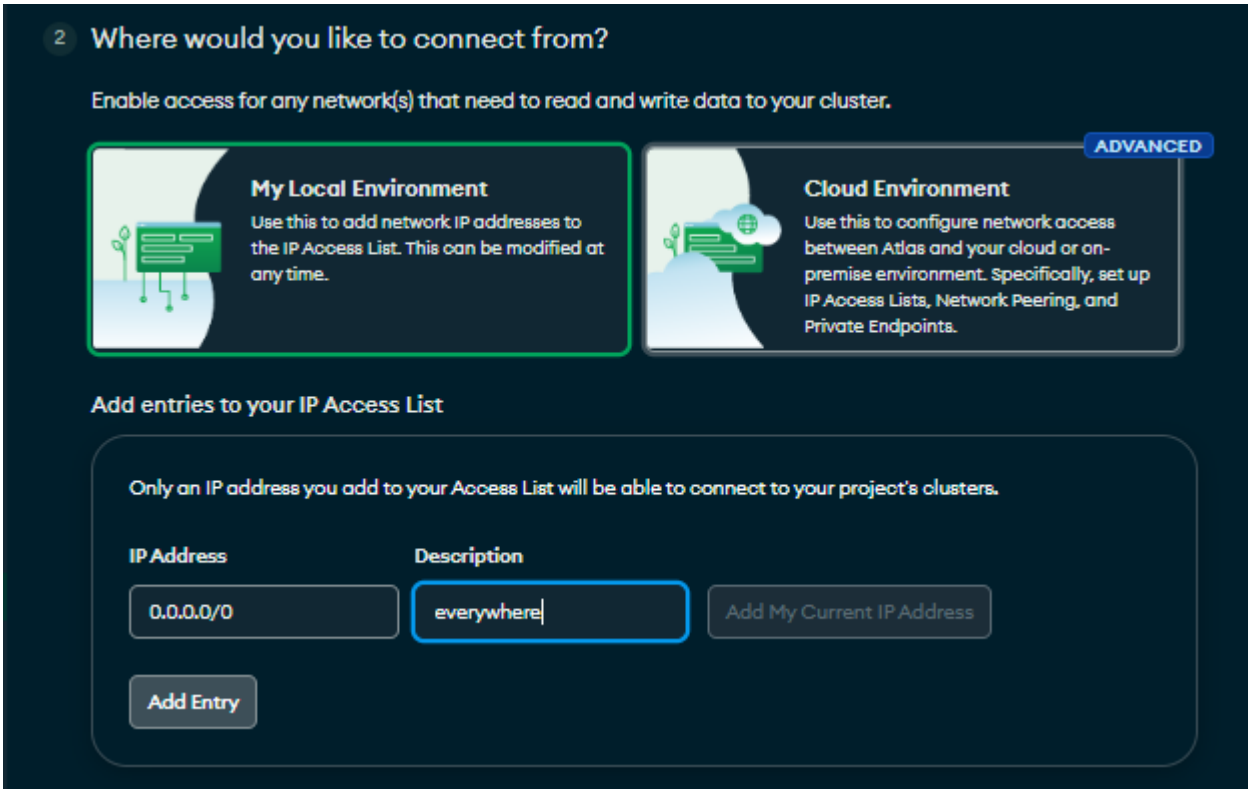


5. Choose a Cluster Configuration:
   MongoDB Atlas allows you to choose your cluster configuration, including the cloud provider, region, and other settings. Select the options that best suit your needs.
   And then click create button.



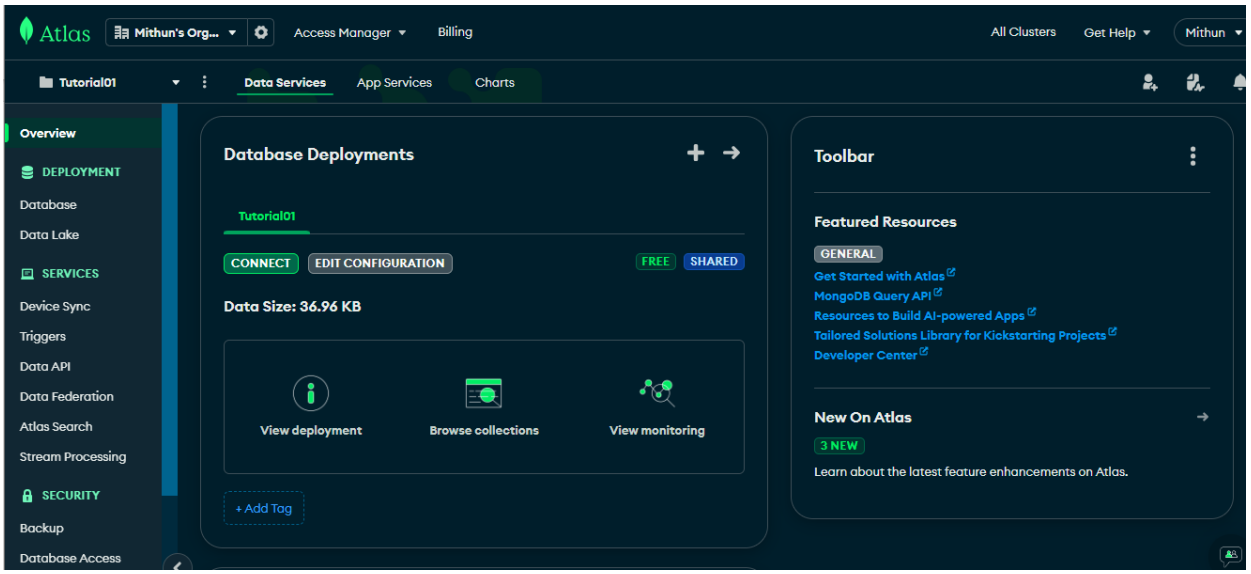
6. Configure Security Settings:
   - Select Username and password as the Authentication method.
   - Provide a username and a strong password. When entering the password, you can use the Autogenerated function. Make sure to copy the password, as it will be required when connecting to the database.



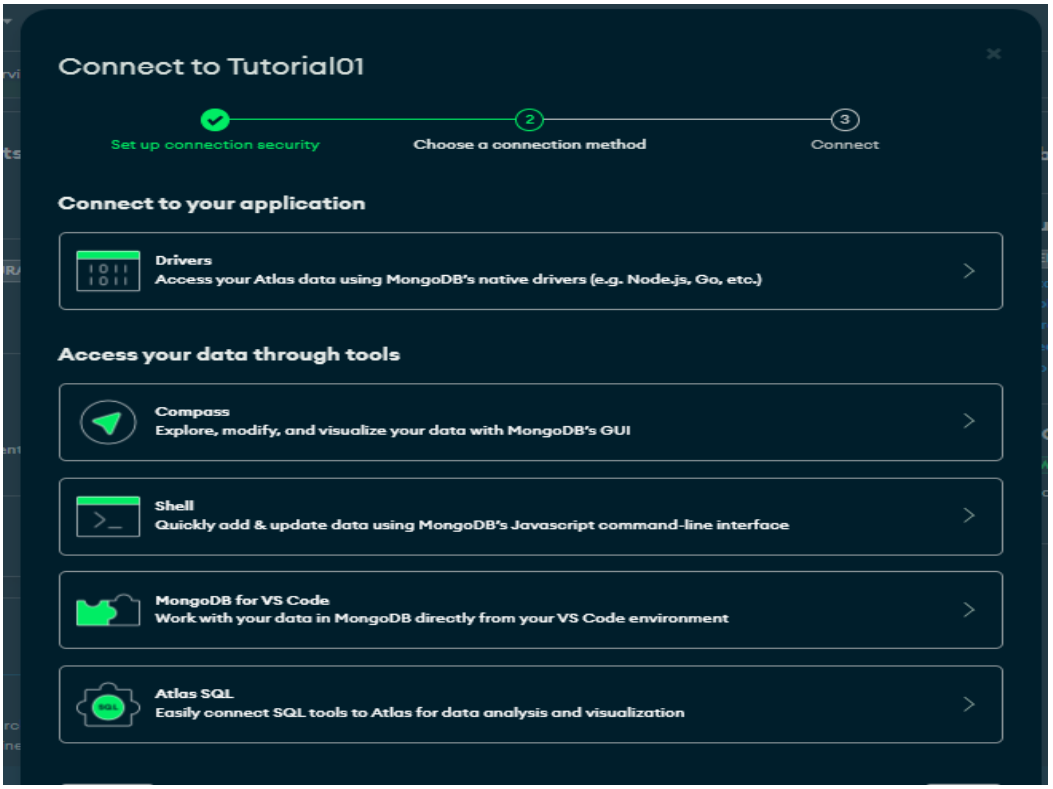


- After providing IP address and Description click the button "finish and close".

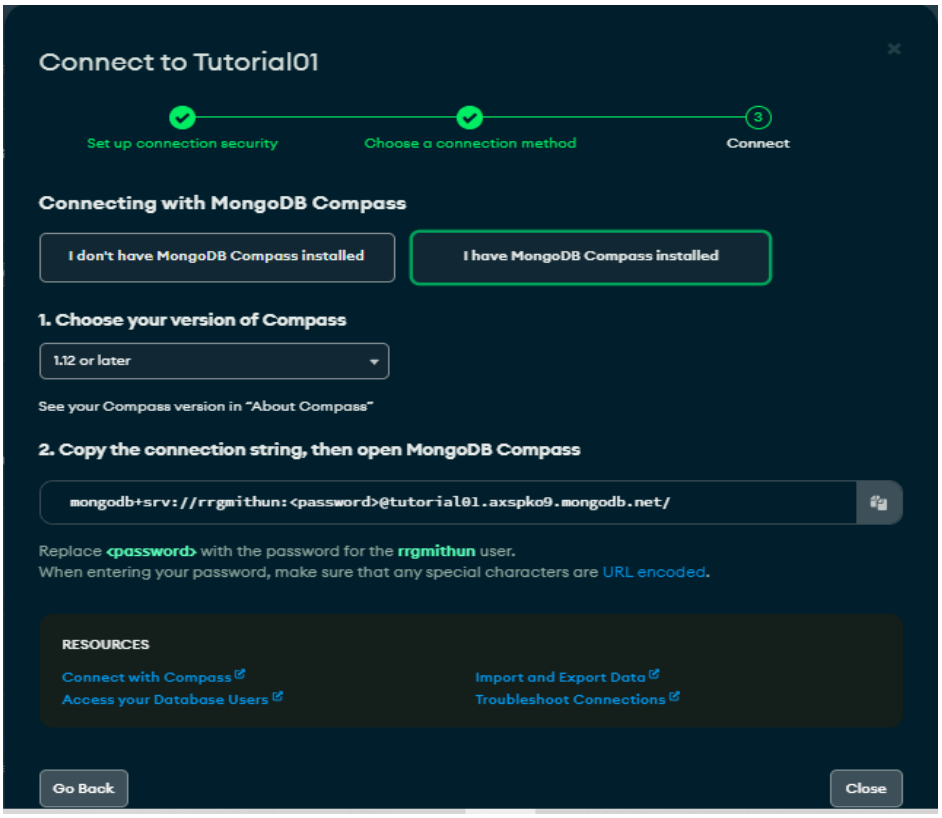
7. Then you must select the created cluster and click the connect button appeared in below Window.



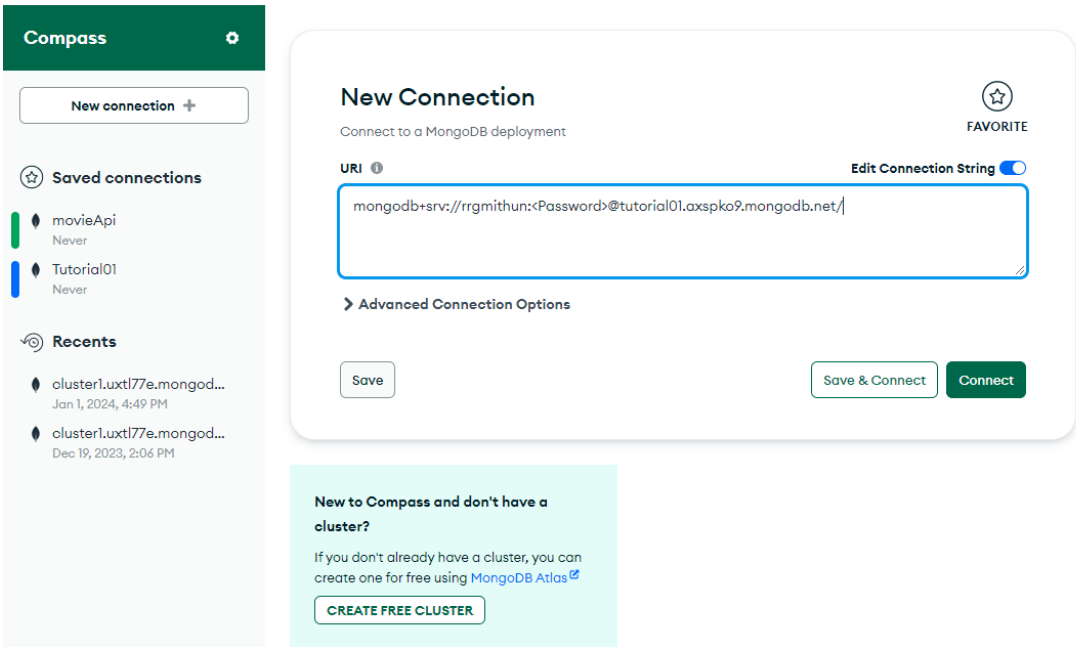
8. When you click the 'Connect' button, a window will appear as shown below. You may encounter different options. For this tutorial, we are using Compass, which features a user-friendly interface (UI).



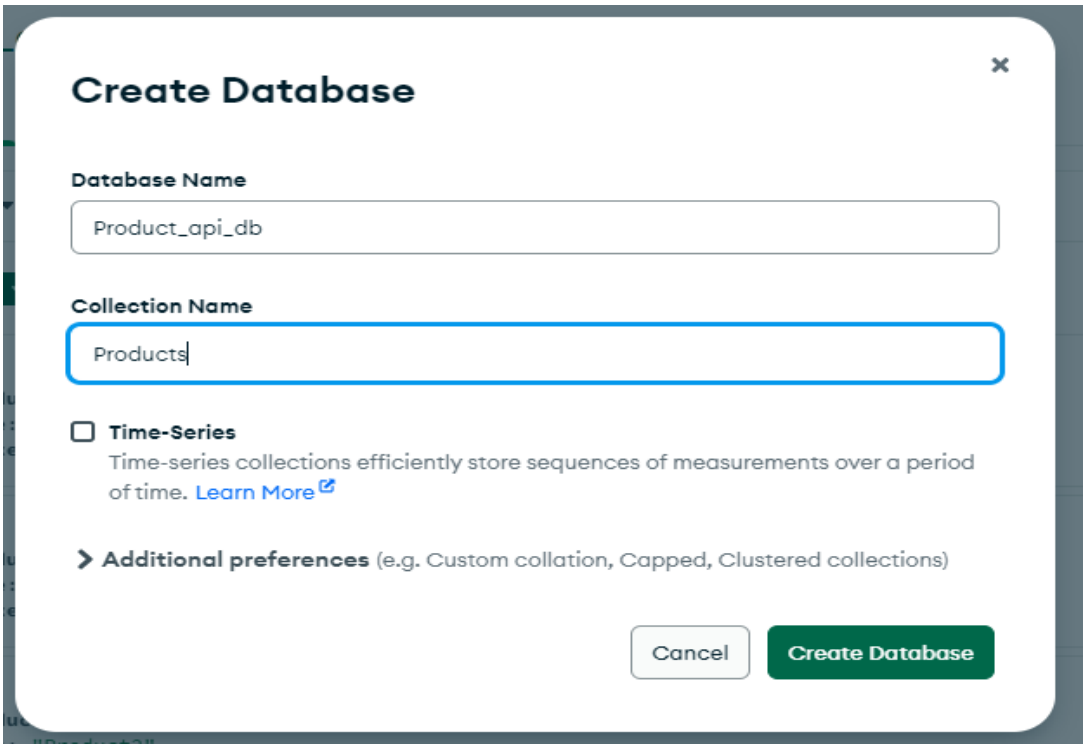
9. After select the Compass you will direct to a window like below. You have to copy the given connection string.



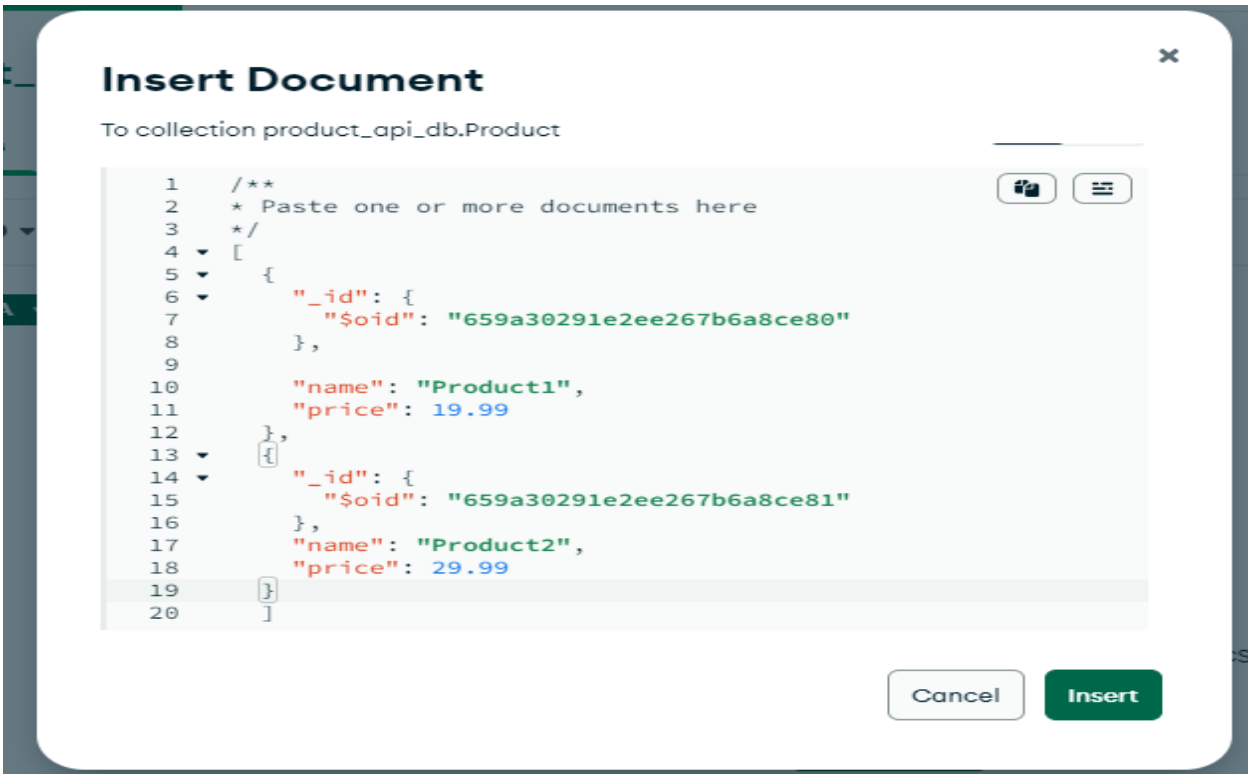
10. Open your MongoDB Compass app in your machine. Click on "New connection" button and paste your connection string in field provide as URI.
Provide your password.
Then click 'connect' button.



11. Click on the '+' appeared in top left corner o add a new database. Then you will appear a window as below. Give database name and collection name as below. The click on the "Create Database" button.

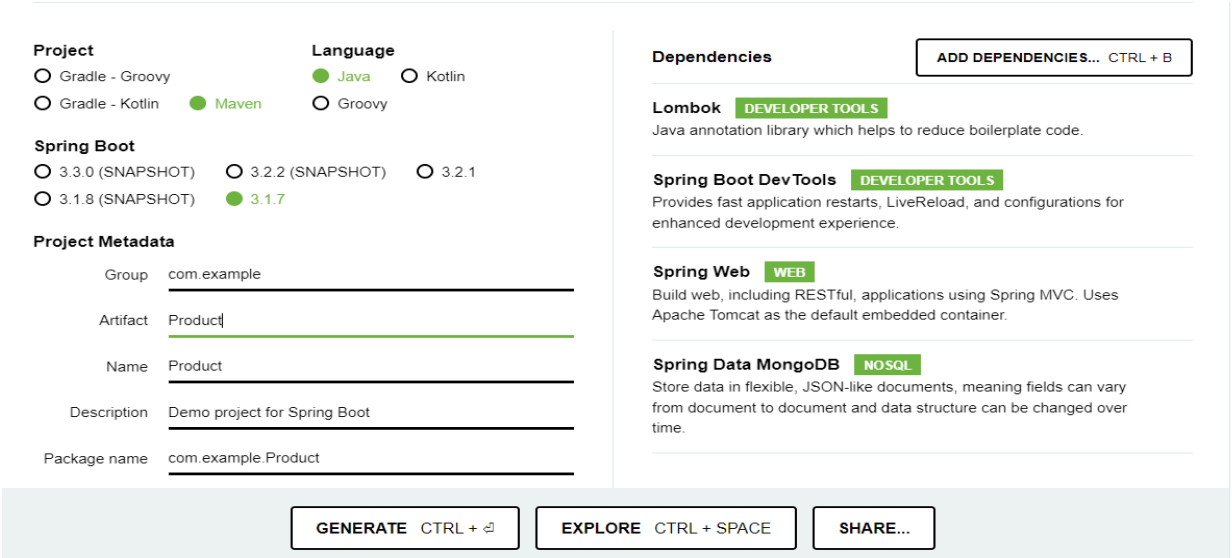


12. Then you can add data by clicking the ADD DATA button. You have give two options, import json or csv file and insert document. Follow window demonstrate how insert a document.



Step 02: Build a Spring boot project.

1. Build your project using Spring Initializr ,name it 'Products' and add the dependencies as shown in the below window.



2. Then open it in IntelliJIdea .
3. Create a new file called .env in the your project directory.

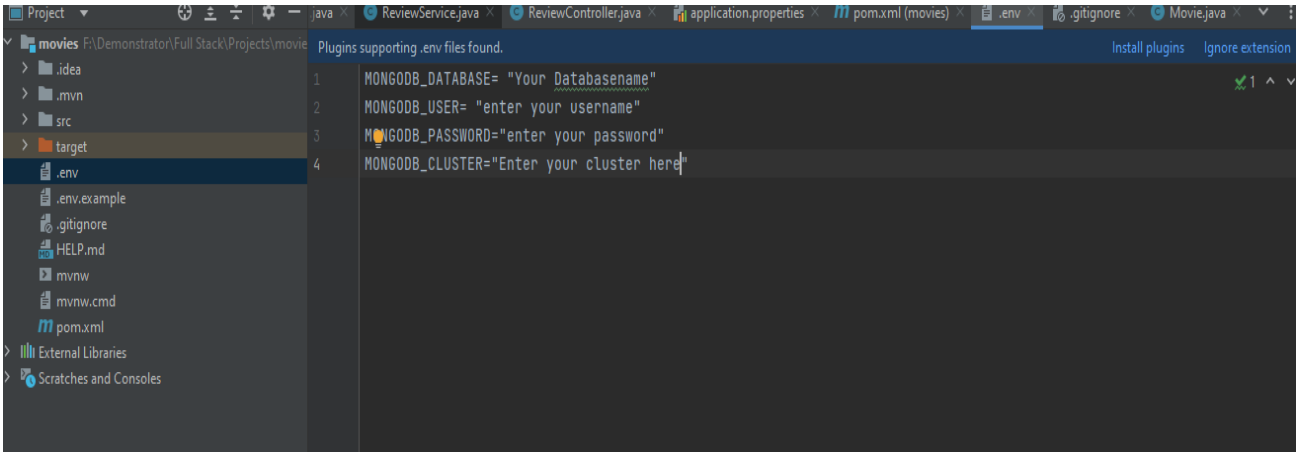
Note:
Connection String consist with following.

mongodb+srv://rrgmithun:**********@tutorial01.lotvyjp.mongodb.net/

Username

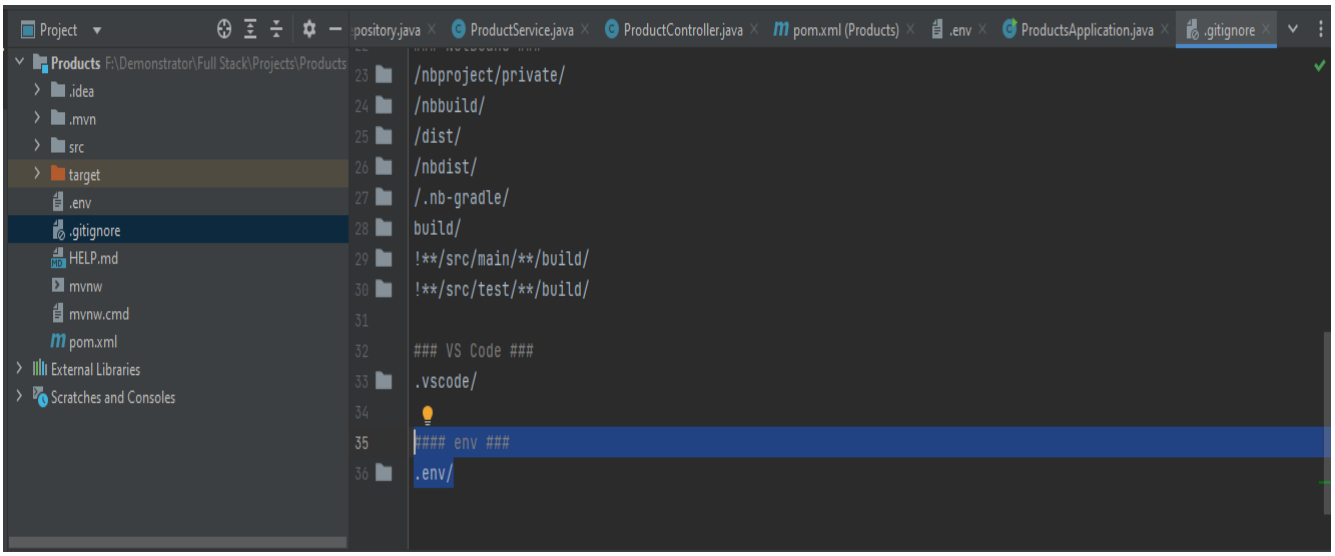
Password

Mongodb cluster

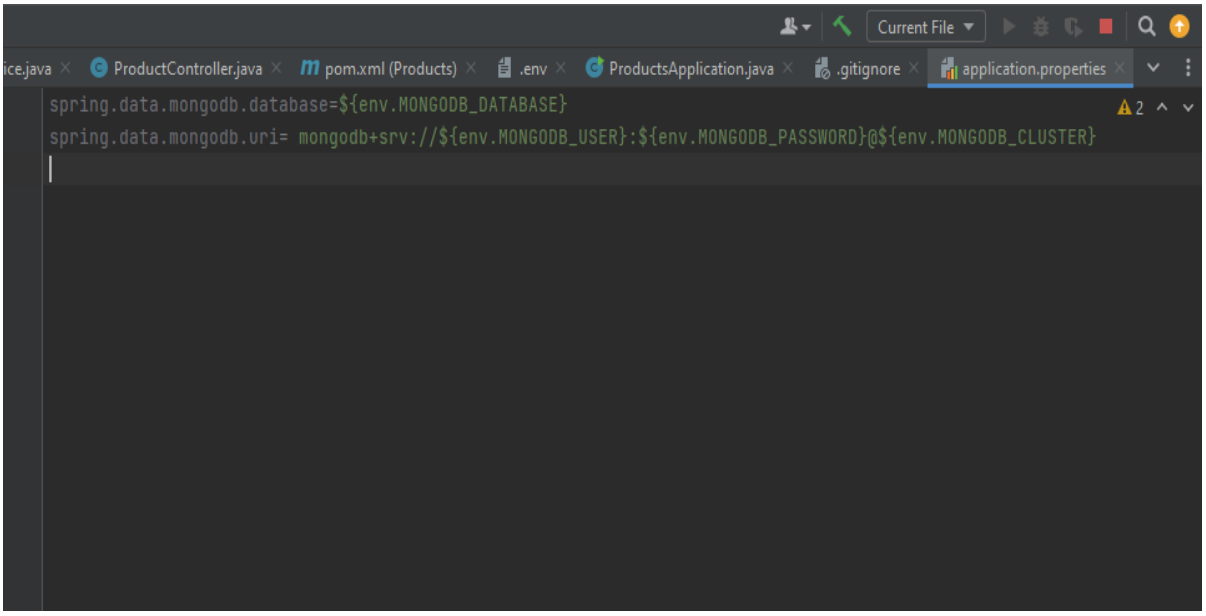


4. Then open your .gitignore file and paste the below given code in it.

```
#### env ###
.env/
```



5. Then go to the directory src/main/resources/application.properties . Add following code to the application.proprtities file.




```
spring.data.mongodb.database=${env.MONGODB_DATABASE}
spring.data.mongodb.uri=
mongodb+srv://${env.MONGODB_USER}:${env.MONGODB_PASSWORD}@${env.MONGOD
B_CLUSTER}
```

6. Then create the product class.

```java
package com.example.Products;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import org.bson.types.ObjectId;
import org.springframework.data.mongodb.core.mapping.Document;
20 usages
@Data
@AllArgsConstructor
@NoArgsConstructor

@Document(collection= "Products")
public class Products {
    private ObjectId id;
    2 usages
    private String productID;
    private String name;
    private double price;

    public String setId(String id) {
        return this. productID;
    }
    public String getId() {
        return productID;
    }
}
```

7. Create ProductRepository interface as below.

```java
package com.example.Products;

import org.bson.types.ObjectId;
import org.springframework.data.mongodb.repository.DeleteQuery;
import org.springframework.data.mongodb.repository.MongoRepository;
import org.springframework.stereotype.Repository;

import java.util.Optional;
1 usage
@Repository
public interface ProductRepository extends  MongoRepository<Products, ObjectId>{

    2 usages
    Optional<Products> findProductsByProductID(String productId);

    1 usage
    @DeleteQuery(value = "{ 'productID' : ?0 }")
    void deleteProductsByProductID(String productId);
}
```

`

8. Create ProductService class as below.

```
package com.example.Products;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import java.util.List;
import java.util.Optional;
@Service
public class ProductService {
        @Autowired
        private ProductRepository productRepository;
        public List<Products> allProducts(){
            return productRepository.findAll();
        }
        public Optional<Products> singleProduct(String productId){
            return productRepository.findProductsByProductID(productId);
        }
    public Products addProduct(Products product) {
        return productRepository.save(product);
    }
    public Products updateProduct(String productId, Products updatedProduct)
{
        Optional<Products> existingProduct =
productRepository.findProductsByProductID(productId);
        if (existingProduct.isPresent()) {
            Products productToUpdate = existingProduct.get();
            productToUpdate.setPrice(updatedProduct.getPrice());
            productToUpdate.setName(updatedProduct.getName());
            // Add other fields that you want to update
            return productRepository.save(productToUpdate);
        } else {
            throw new RuntimeException("Product with ID " + productId + " not
found");
        }
    }
    public void deleteProduct(String productId) {
        productRepository.deleteProductsByProductID(productId);
    }
    }
```

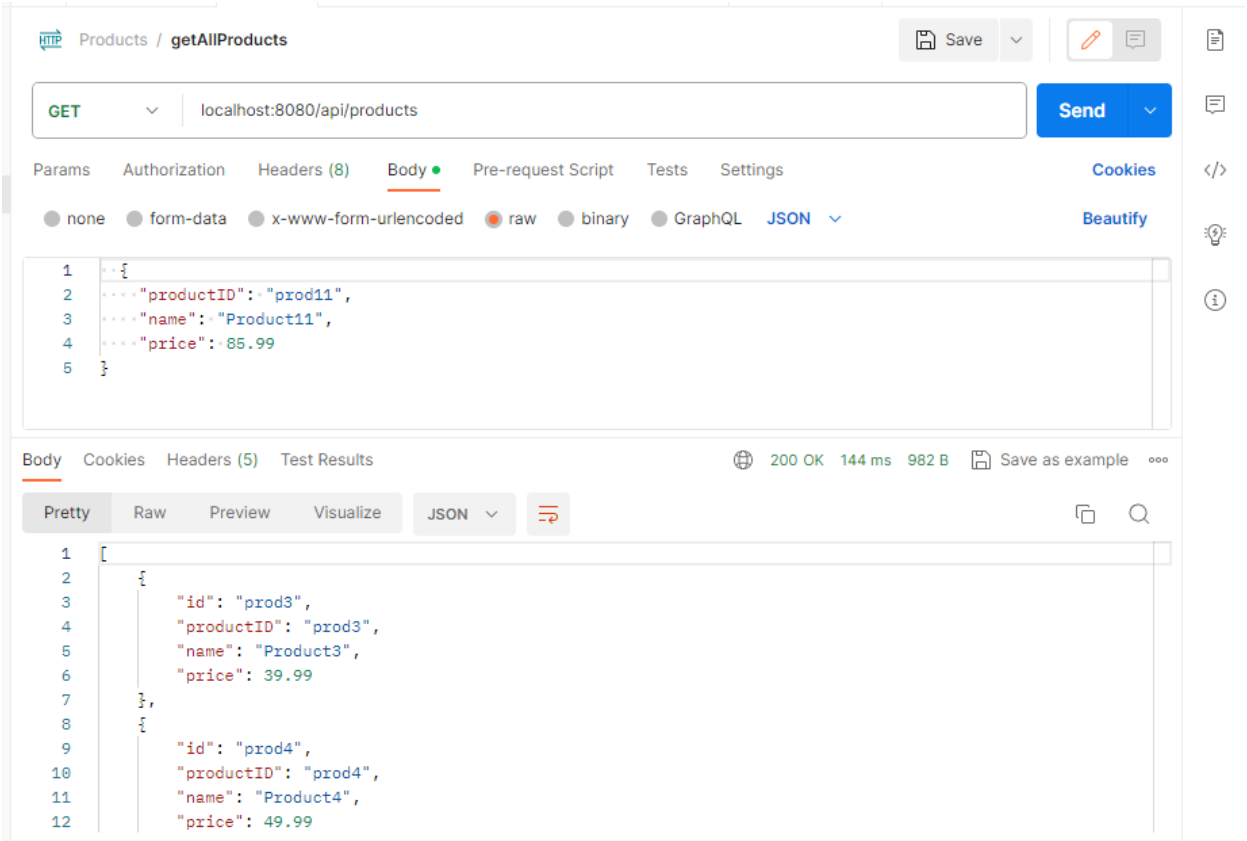
9. Create productController class as below.

```
package com.example.Products;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.List;
import java.util.Optional;


@RestController
@RequestMapping("/api/products")
public class ProductController {


    @Autowired
    private ProductService productService;
    @GetMapping
    public ResponseEntity<List<Products>> getAllProducts(){
        return new
ResponseEntity<List<Products>>(productService.allProducts(), HttpStatus.OK);
    }
    @GetMapping("/{productId}")
    public ResponseEntity<Optional<Products>> getSingleProduct(@PathVariable
String productId ){
        return new ResponseEntity<Optional<Products>>
(productService.singleProduct(productId),HttpStatus.OK );
    }
    @PostMapping
    public ResponseEntity<Products> addProduct(@RequestBody Products product)
{
        Products savedProduct = productService.addProduct(product);
        return new ResponseEntity<>(savedProduct, HttpStatus.CREATED);
    }

    @PutMapping("/{productId}")
    public ResponseEntity<Products> updateProduct(@PathVariable String
productId, @RequestBody Products updatedProduct) {
        Products updated = productService.updateProduct(productId,
updatedProduct);
        return new ResponseEntity<>(updated, HttpStatus.OK);
    }

    @DeleteMapping("/{productId}")
    public ResponseEntity<Void> deleteProduct(@PathVariable String productId)
{
        productService.deleteProduct(productId);
        return new ResponseEntity<>(HttpStatus.NO_CONTENT);
    }
}
```

Step 03: Testing the rest api by Postman.

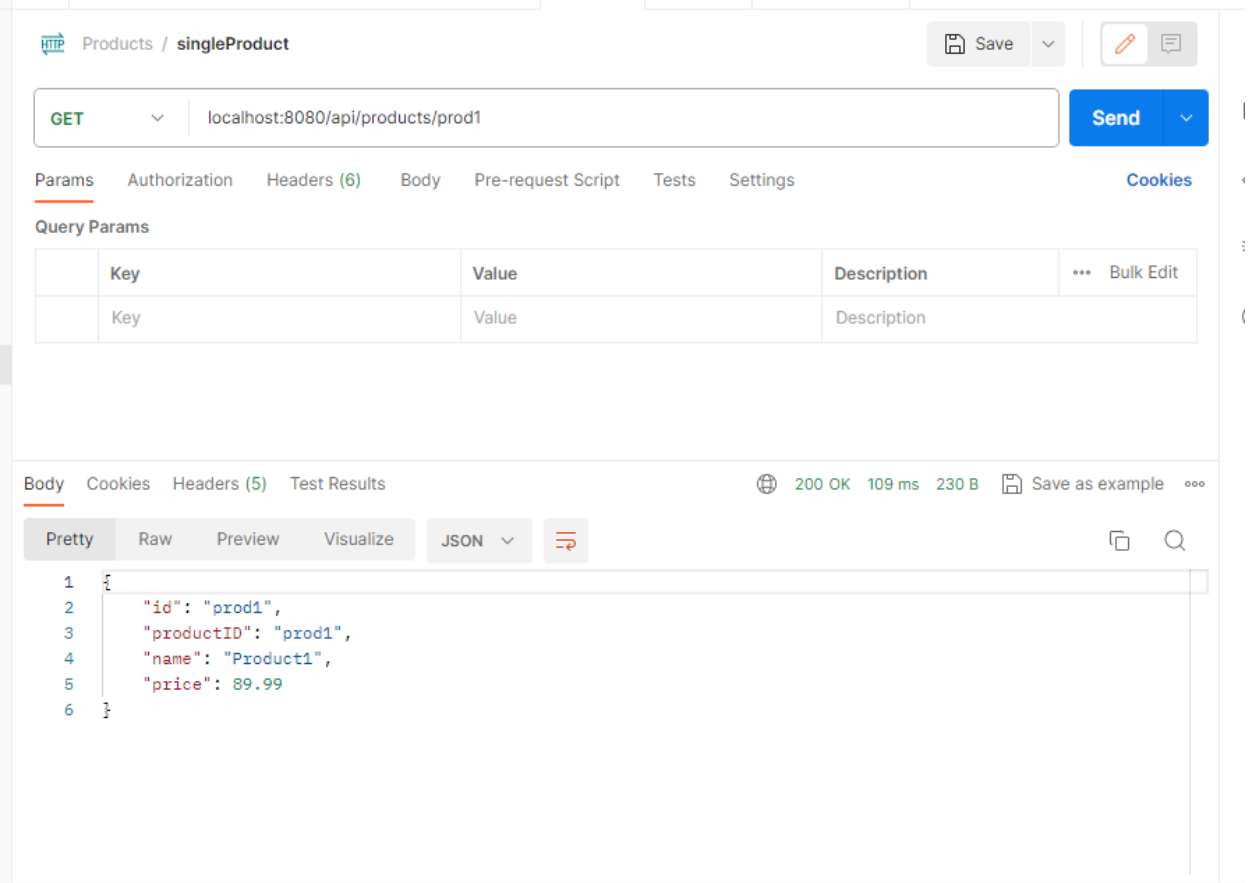
1. Get All Products

- Endpoint: GET /api/products
- Explanation: Retrieves a list of all products.
- Testin in Postman:
    - Set the request type to GET.
    - Enter the URL: http://localhost:8080/api/products .
    - Click the "Send" button.



2. Get single product

- Endpoint: GET /api/products/{productId}
- Explanation: Retrieves information about a specific product based on its ID.
- Testin in Postman:
    - Set the request type to GET.
    - Enter the URL, for example: http://localhost:8080/api/products/yourProductId .
    - Replace yourProductId with an actual product ID.
    - Click the "Send" button.

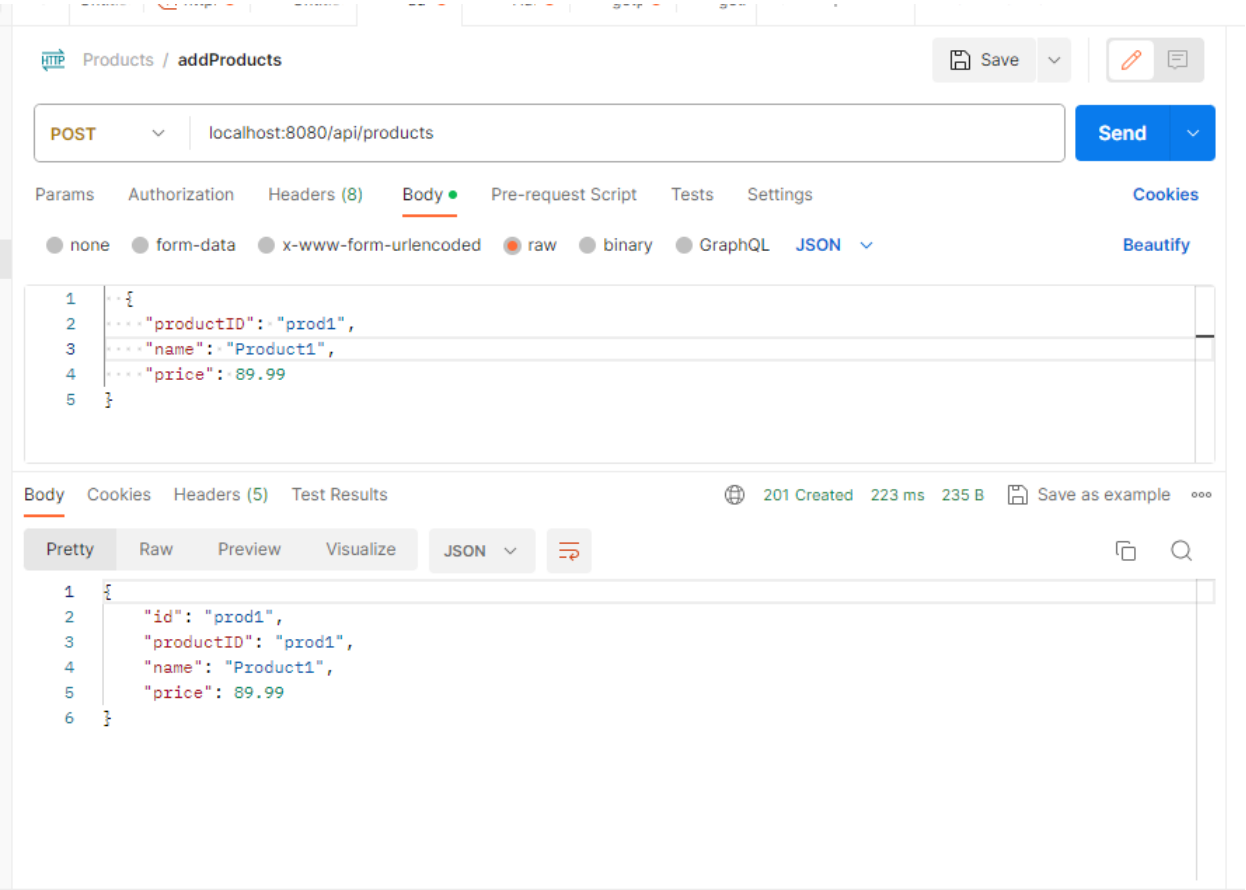


3. Add product

- Endpoint: POST ‘/api/products’
- Explanation: Adds a new product to the system.
- Testin in Postman:
    - Set the request type to POST.
    - Enter the URL: http://localhost:8080/api/products.
    - Go to the "Body" tab, select "raw," and choose "JSON (application/json)" from the dropdown.
    - Enter the product details in JSON format. For example:
    -

```
{
 "productID": "newProduct",
 "name": "New Product",
 "price": 39.99
}
```

    - Click the "Send" button.

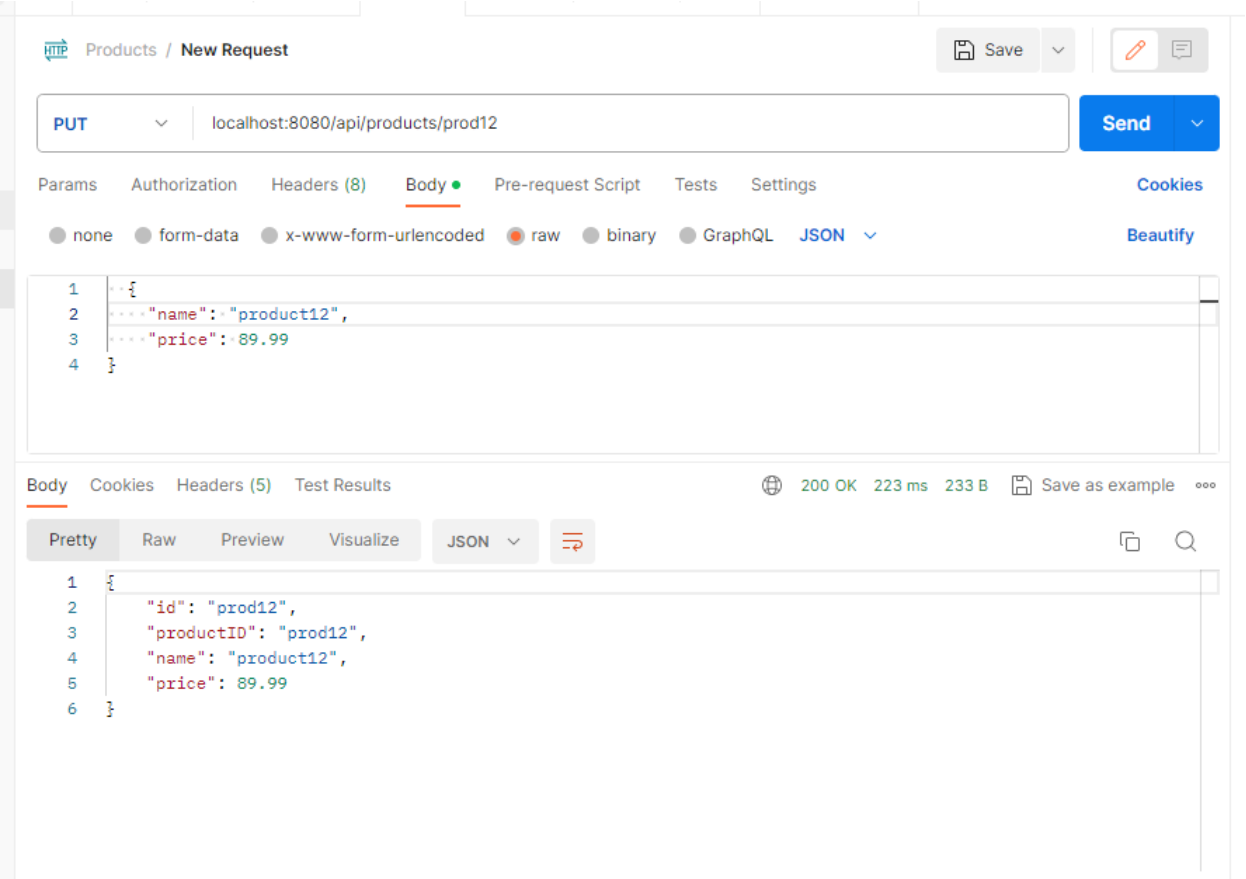


4. Update Products:

- Endpoint: PUT /api/products/{productId}
- Explanation Updates the information of an existing product based on its ID.
- Testin in Postman:
    - Set the request type to PUT.
    - Enter the URL: http://localhost:8080/api/products/yourProductId .
    - Replace yourProductId with an actual product ID.
    - Go to the "Body" tab, select "raw," and choose "JSON (application/json)" from the dropdown.
    - Enter the fields you want to update in JSON format. For example, to update the name and price:

```
{
  "name": "New Product",
  "price": 39.99
}
```

    - Click the "Send" button.



5. Delete products:

- Endpoint: DELETE **/api/products/{productId}**
- Explanation Deletes a product based on its ID.
- Testin in Postman:
    - Set the request type to DELETE
    - Enter the URL, for example: http://localhost:8080/api/products/yourProductId.
    - Replace yourProductId with an actual product ID.
    - Click the "Send" button.