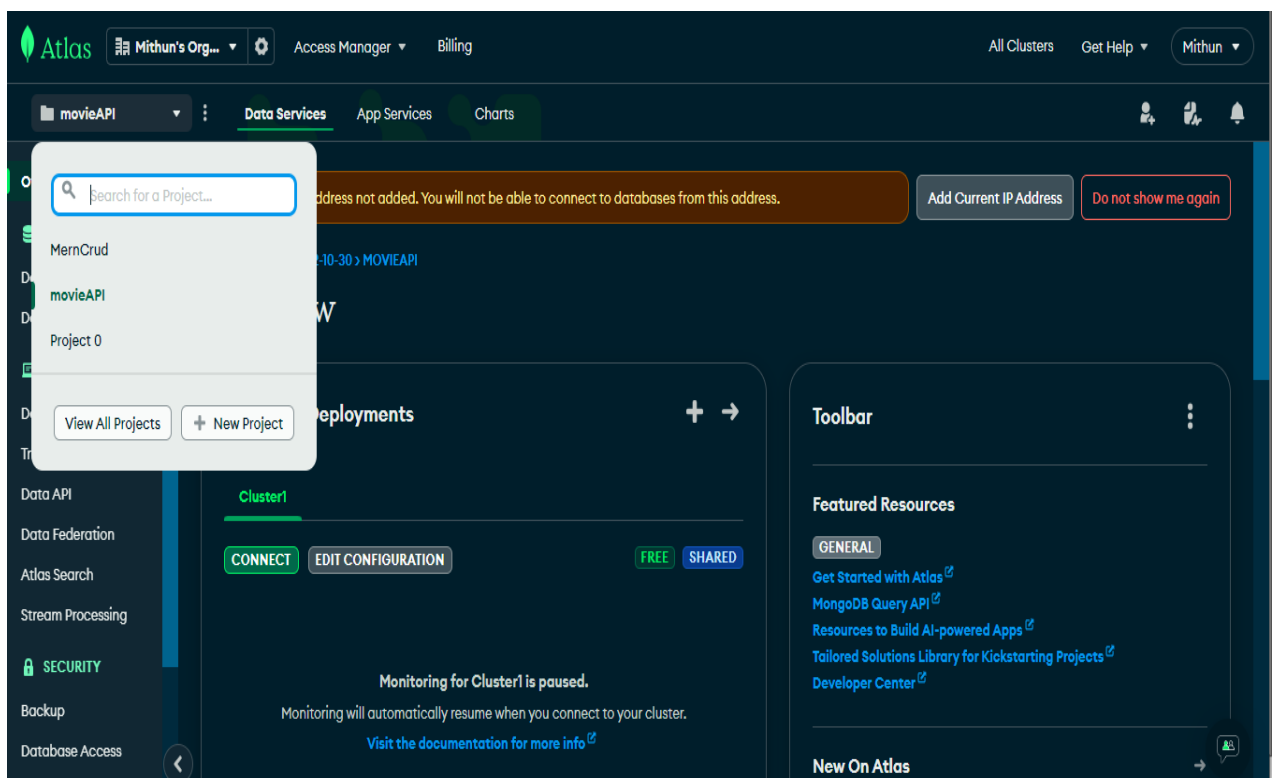




Department of Statistics & Computer Science
University of Kelaniya
ACADEMIC YEAR – 2021/2022
COSC 32133
Full-Stack Software Development
Practical Tutorial 03

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.

MongoDB

Deploy your database

Use a template below or set up [advanced configuration options](#). You can also edit these configuration options once the cluster is created.

M10 **\$0.08/hour**
For production applications with sophisticated workload requirements.

STORAGE	RAM	vCPU
10 GB	2 GB	2 vCPUs

SERVERLESS **\$0.10/1M reads**
For application development and testing, or workloads with variable traffic.

STORAGE	RAM	vCPU
Up to 1 TB	Auto-scale	Auto-scale

M0 **FREE**
For learning and exploring MongoDB in a cloud environment.

STORAGE	RAM	vCPU
512 MB	Shared	Shared

Provider

☒ AWS ☐ Google Cloud ☐ Azure

Region

☒ Mumbai (ap-south-1) ☐ ☐ ☐

★ Recommended ☐ Low carbon emissions ☐

Name

You cannot change the name once the cluster is created.

Tag (optional)

Create your first tag to categorize and label your resources; more tags can be added later. [Learn more.](#)

:

FREE

[Create](#)

Free forever! Your M0 cluster is ideal for experimenting in a limited credential. You can upgrade to a production cluster anytime.

[Access Advanced Configuration](#)

[I'll deploy my database later](#)

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.

Security Quickstart

To access data stored in Atlas, you'll need to create users and set up network security controls. [Learn more about security](#)

✓ How would you like to authenticate your connection?


Your first user will have permission to read and write any data in your project.


Username and Password


Certificate


Create a database user using a username and password. Users will be given the *read and write to any database* privilege by default. You can update these permissions and/or create additional users later. Ensure these credentials are different to your MongoDB Cloud username and password. You can manage existing users via the [Database Access Page](#).

Username

Password 


 Autogenerate Secure Password

 Copy

finished provisioning. 


2 Where would you like to connect from?

Enable access for any network(s) that need to read and write data to your cluster.



My Local Environment

Use this to add network IP addresses to the IP Access List. This can be modified at any time.



Cloud Environment

Use this to configure network access between Atlas and your cloud or on-premise environment. Specifically, set up IP Access Lists, Network Peering, and Private Endpoints.

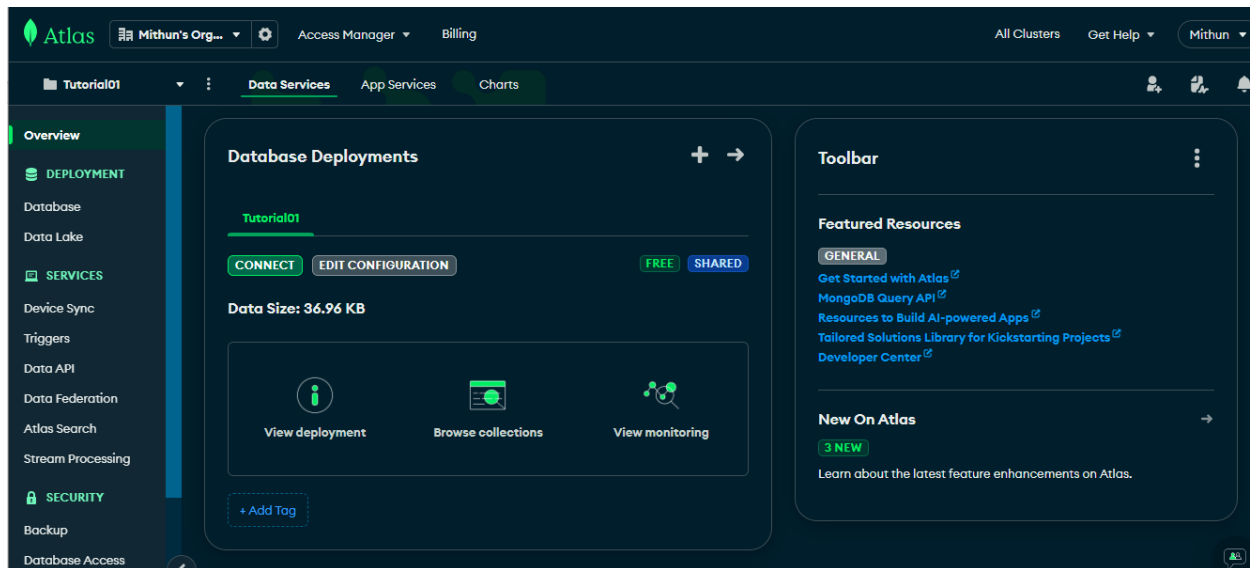
ADVANCED

Add entries to your IP Access List

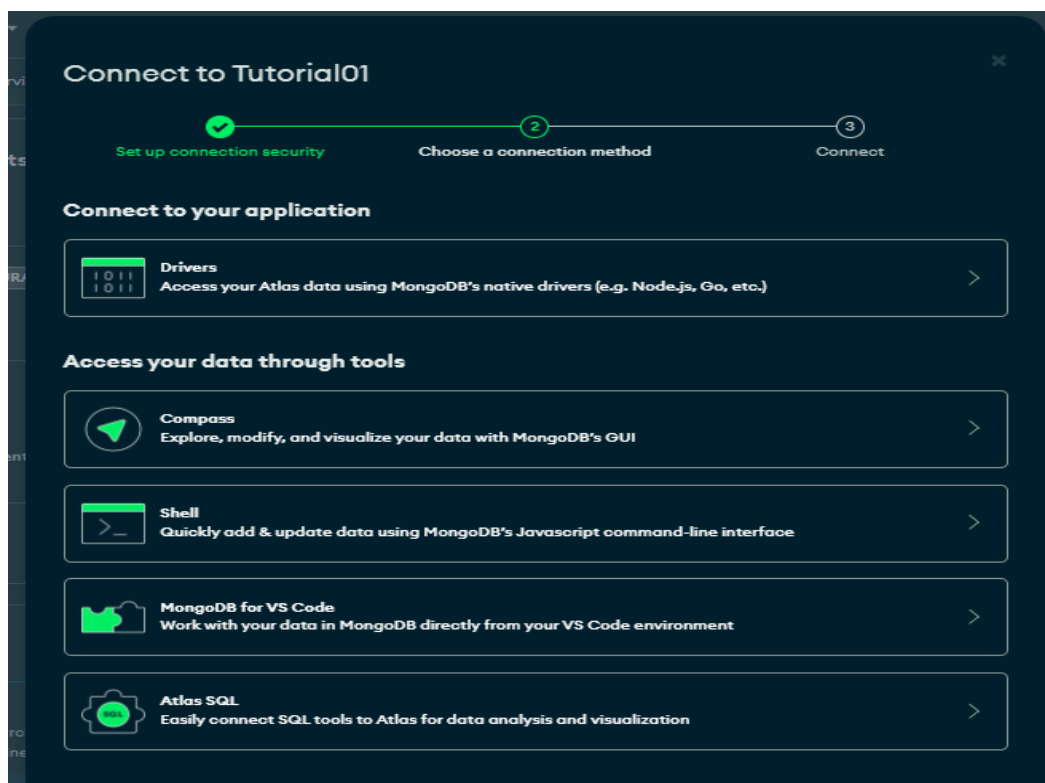
Only an IP address you add to your Access List will be able to connect to your project's clusters.

IPAddress	Description	
<input type="text" value="0.0.0.0/0"/>	<input type="text" value="everywhere"/>	<input type="button" value="Add My Current IP Address"/>
<input type="button" value="Add Entry"/>		

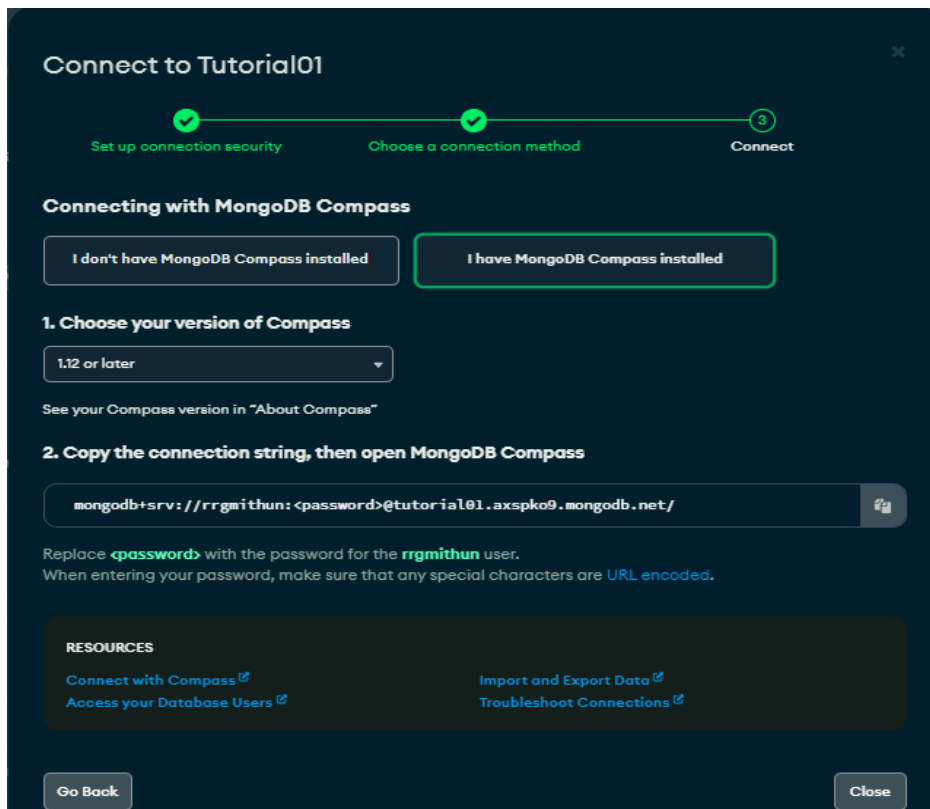
- After providing IP address and Description click the button “finish and close”.
7. Then you must select the created cluster and click the connect button appeared in below Window.



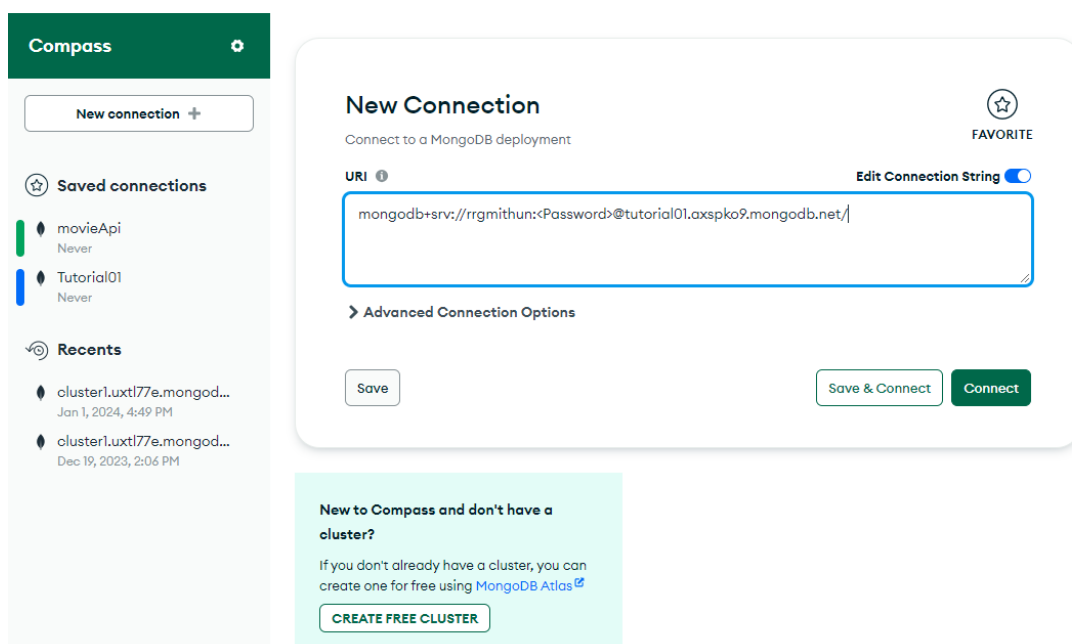
8. When you click the 'Connect' button, a window will appear as shown below. You may encounter different options. For this tutorial, we are using **Compass**, which features a user-friendly interface (UI).



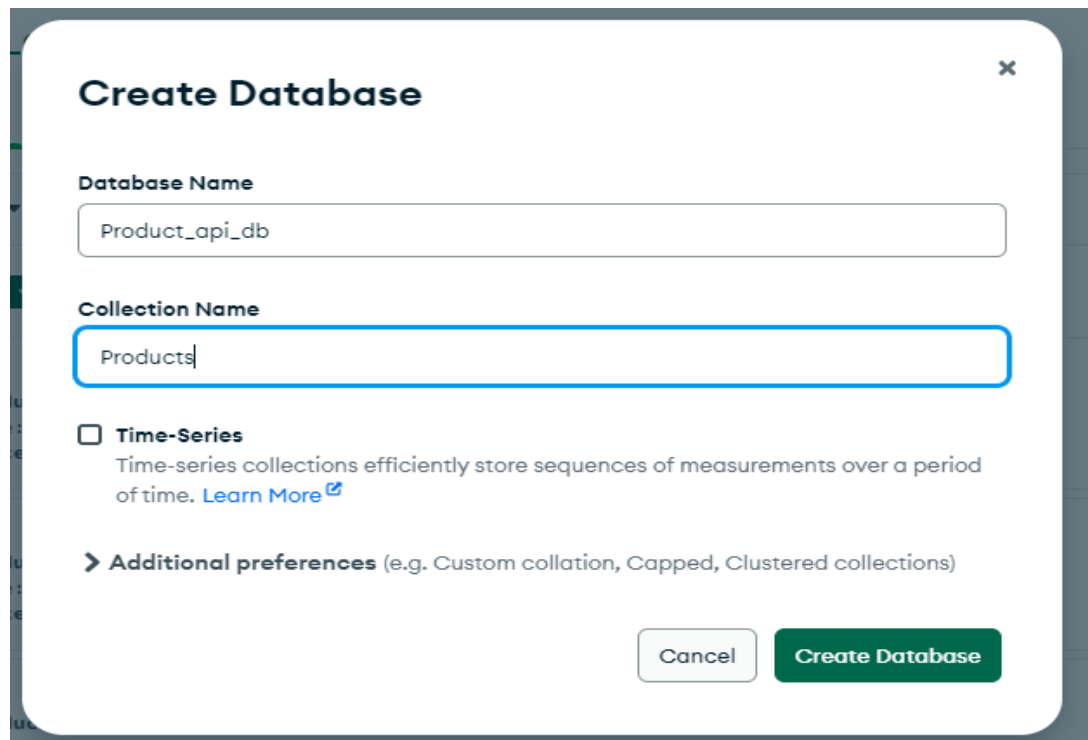
9. After select the Compass you will direct to a window like below. You have to copy the given connection string.



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.



The 'Create Database' dialog box features a title bar with a close button (X). It contains two input fields: 'Database Name' with the value 'Product_api_db' and 'Collection Name' with the value 'Products'. Below these fields is a checkbox for 'Time-Series' with a description and a 'Learn More' link. At the bottom, there is a link for 'Additional preferences' and two buttons: 'Cancel' and 'Create Database'.

Create Database

Database Name
Product_api_db

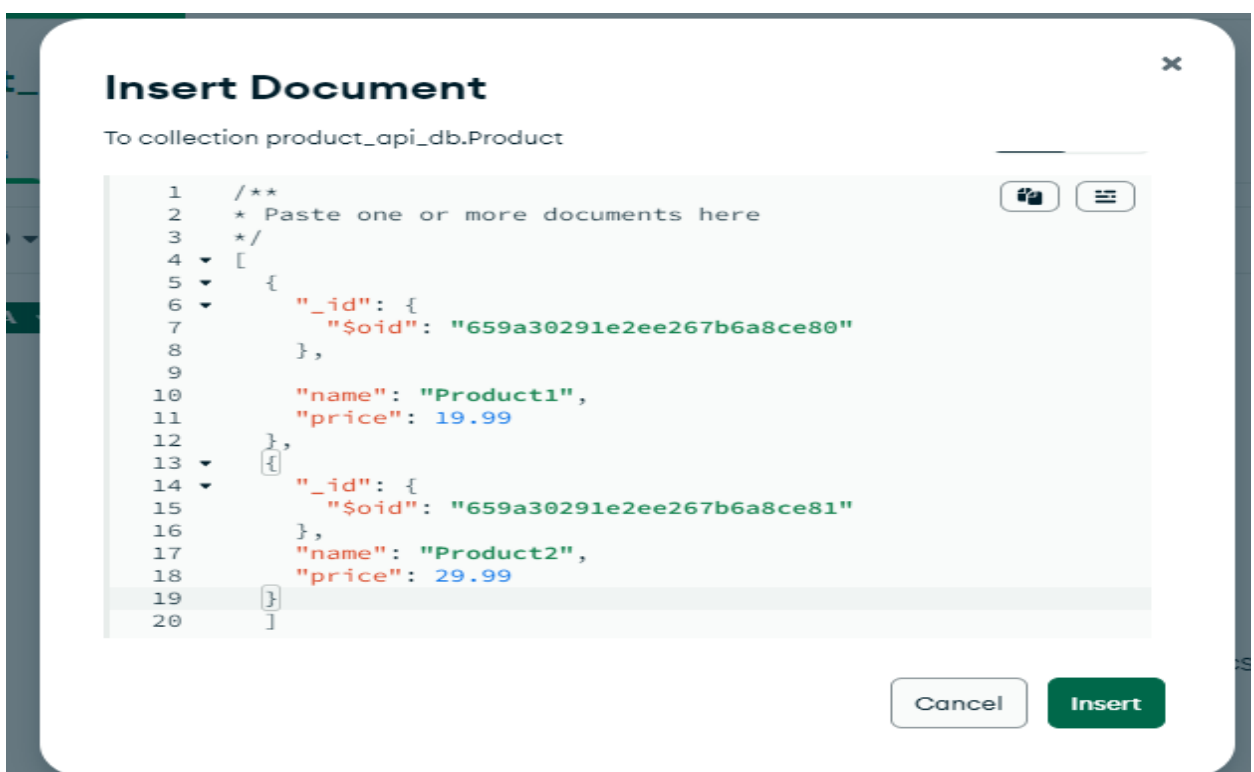
Collection Name
Products

☐ **Time-Series**
Time-series collections efficiently store sequences of measurements over a period of time. [Learn More](#)

[Additional preferences](#) (e.g. Custom collation, Capped, Clustered collections)

Cancel Create Database

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.



The 'Insert Document' dialog box has a title bar with a close button (X). It shows the target collection 'product_api_db.Product'. A text area contains a JSON array of two documents. At the bottom, there are 'Cancel' and 'Insert' buttons.

Insert Document

To collection product_api_db.Product

```
1  /**
2  * Paste one or more documents here
3  */
4  [
5    {
6      "_id": {
7        "$oid": "659a30291e2ee267b6a8ce80"
8      },
9      "name": "Product1",
10     "price": 19.99
11   },
12   {
13     "_id": {
14       "$oid": "659a30291e2ee267b6a8ce81"
15     },
16     "name": "Product2",
17     "price": 29.99
18   }
19 ]
20
```

Cancel Insert

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.

The screenshot shows the Spring Initializr web interface. On the left, under 'Project', 'Gradle - Groovy' is selected. Under 'Language', 'Java' is selected. Under 'Spring Boot', '3.1.7' is selected. The 'Project Metadata' section has 'Group' as 'com.example', 'Artifact' as 'Product', 'Name' as 'Product', 'Description' as 'Demo project for Spring Boot', and 'Package name' as 'com.example.Product'. On the right, the 'Dependencies' section shows 'Lombok' (Developer Tools), 'Spring Boot DevTools' (Developer Tools), 'Spring Web' (Web), and 'Spring Data MongoDB' (NoSQL). At the bottom, there are buttons for 'GENERATE' (CTRL + G), 'EXPLORE' (CTRL + SPACE), and 'SHARE...'.

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://**rgmithun**:*********@**tutorial01.lotvyjp.mongodb.net/**

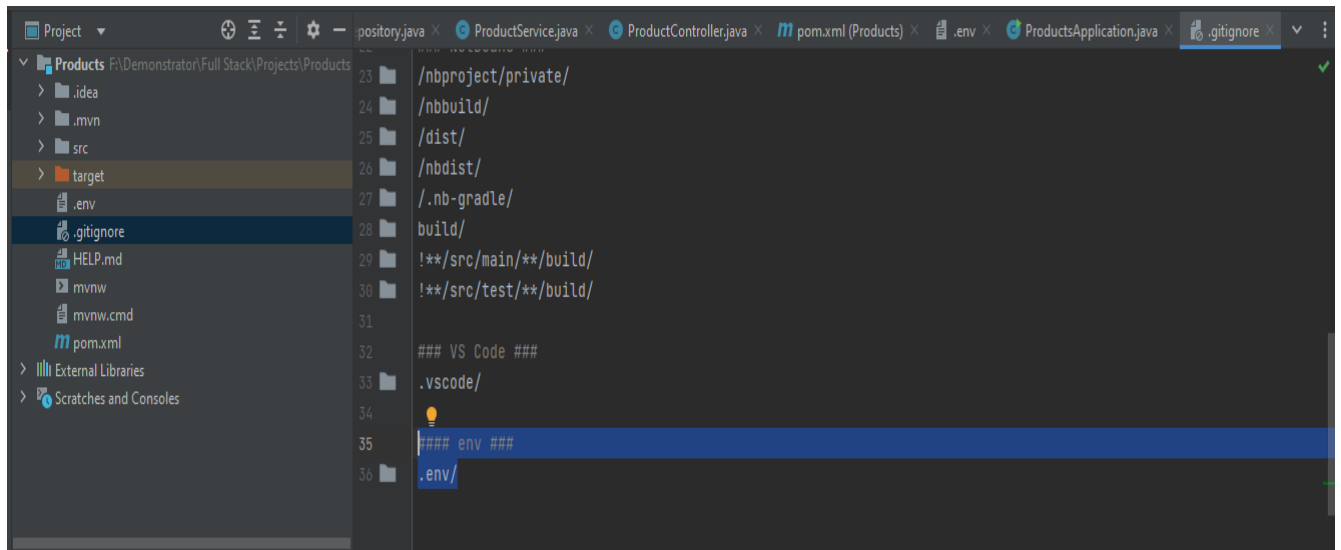
Username	Password	Mongodb cluster
-----------------	-----------------	------------------------

The screenshot shows the IntelliJ IDEA interface with the **.env** file open. The file contains the following content:

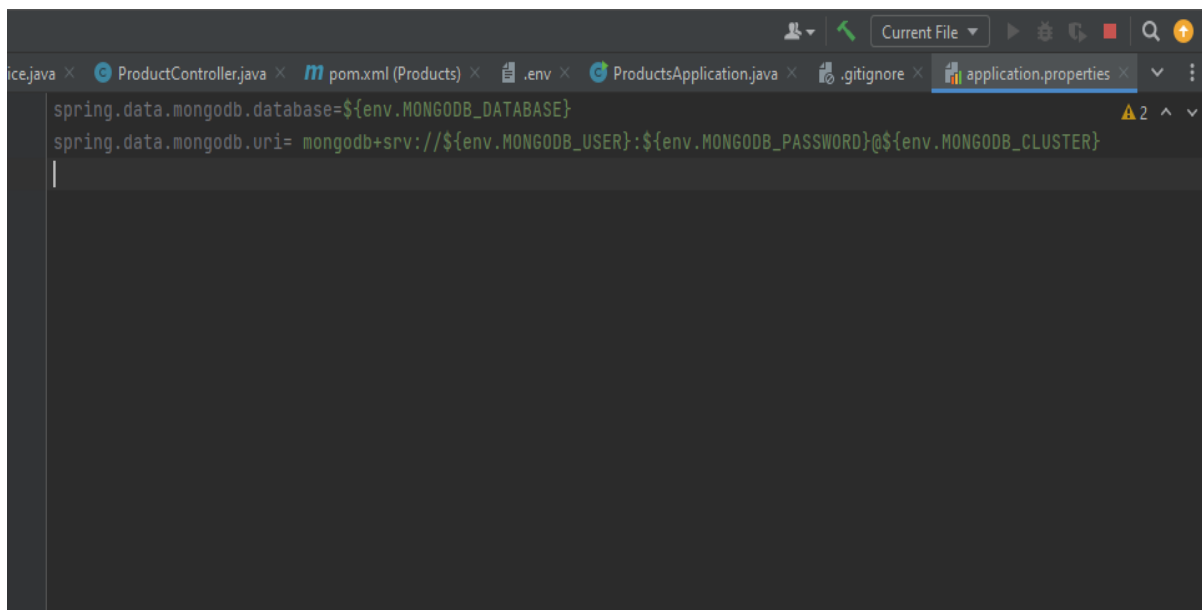
```
1 MONGODB_DATABASE= "Your Databasename"
2 MONGODB_USER= "enter your username"
3 MONGODB_PASSWORD="enter your password"
4 MONGODB_CLUSTER="Enter your cluster here"
```

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.MONGODB_CLUSTER}
```


6. Then create the product class.

```
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.

```
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);
    }

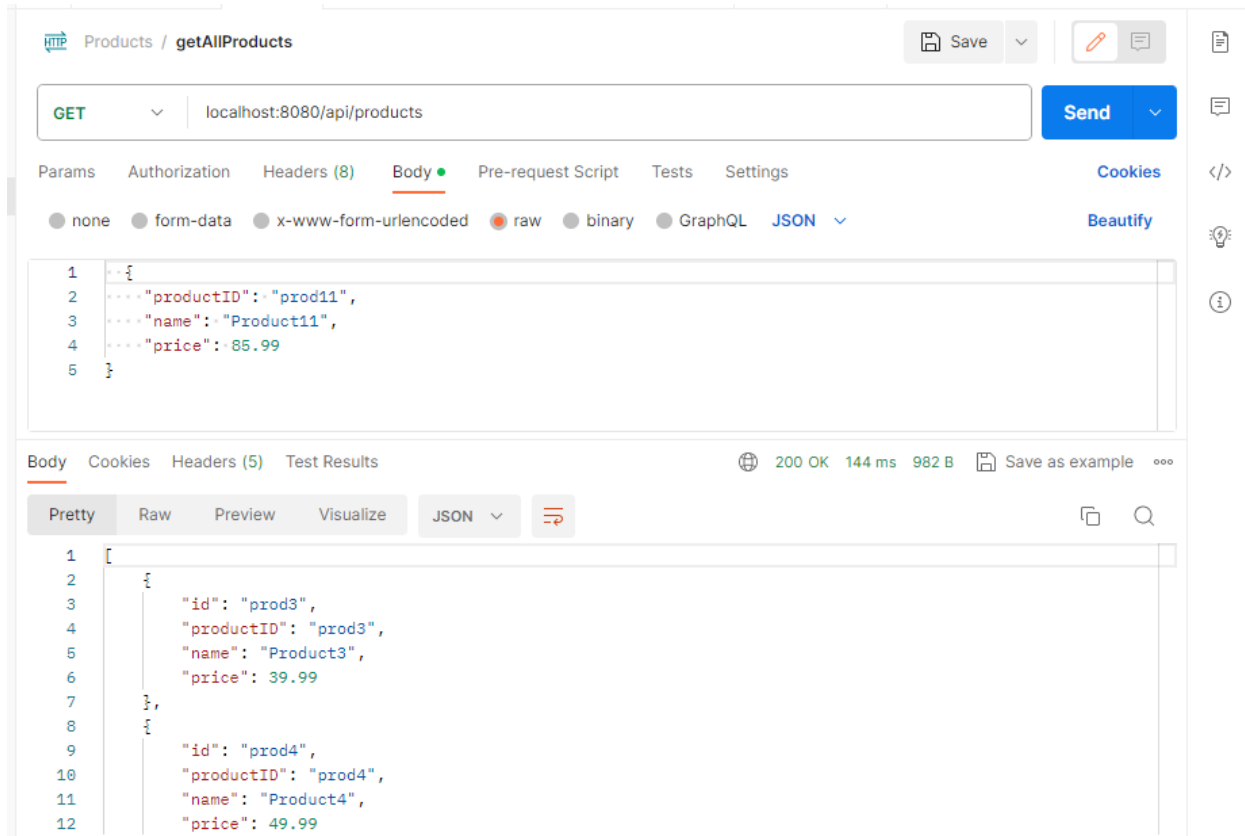
    @PutMapping("/{productId}")
    public ResponseEntity<Products> updateProduct(@PathVariable String
productId, @RequestBody Products updatedProduct) {
        Products updated = productService.updateProduct(productId,
updatedProduct);
        return new ResponseEntity<>(updated, HttpStatus.OK);
    }

    @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.
- Test 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.

The screenshot shows the Postman interface for a GET request to `localhost:8080/api/products/prod1`. The request is saved and has a "Send" button. The "Params" tab is active, showing a table for Query Params.

	Key	Value	Description	...	Bulk Edit
	Key	Value	Description		

The "Body" tab is active, showing the response in JSON format. The response is a 200 OK status with a response time of 109 ms and a body size of 230 B. The JSON data is as follows:

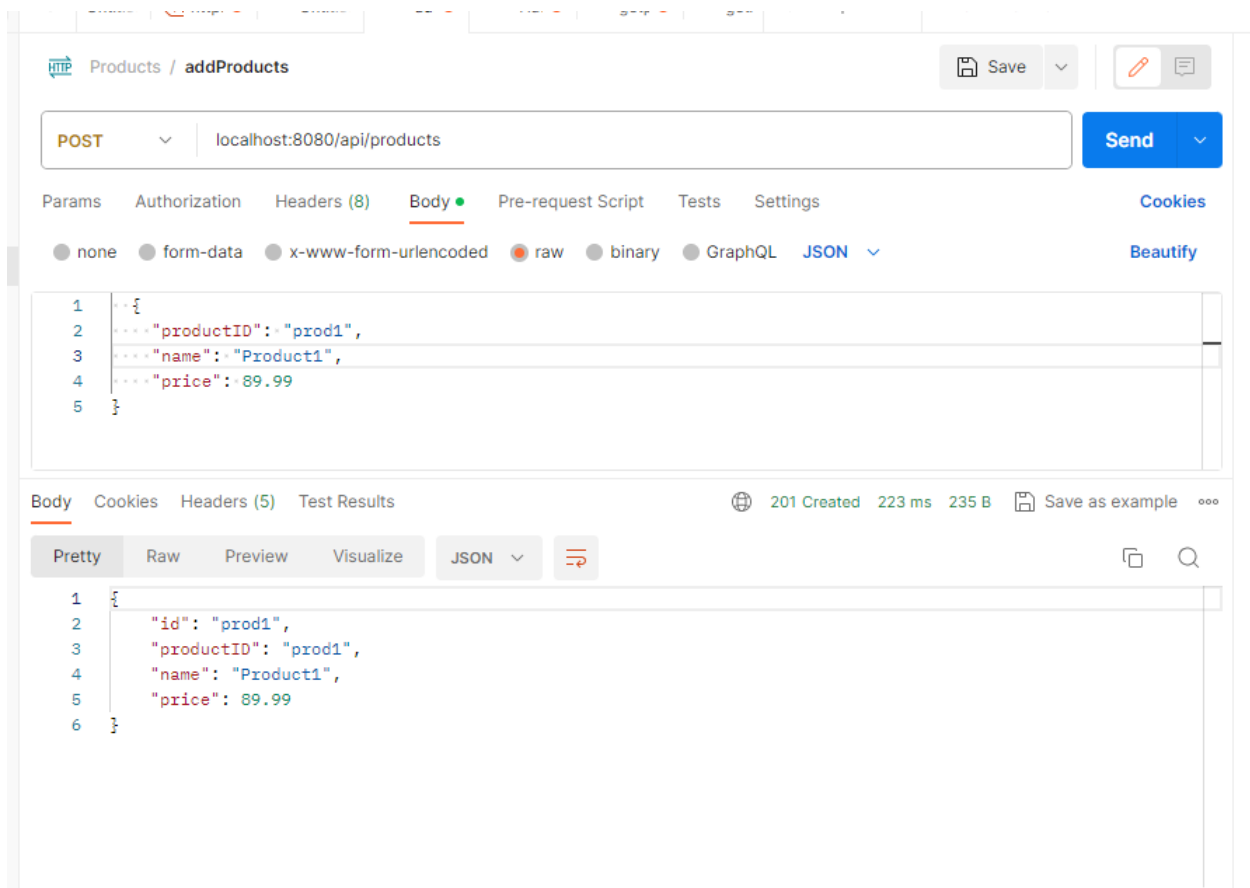
```
1 {
2   "id": "prod1",
3   "productId": "prod1",
4   "name": "Product1",
5   "price": 89.99
6 }
```

3. Add product

- Endpoint: POST '/api/products'
- Explanation: Adds a new product to the system.
- Test 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.

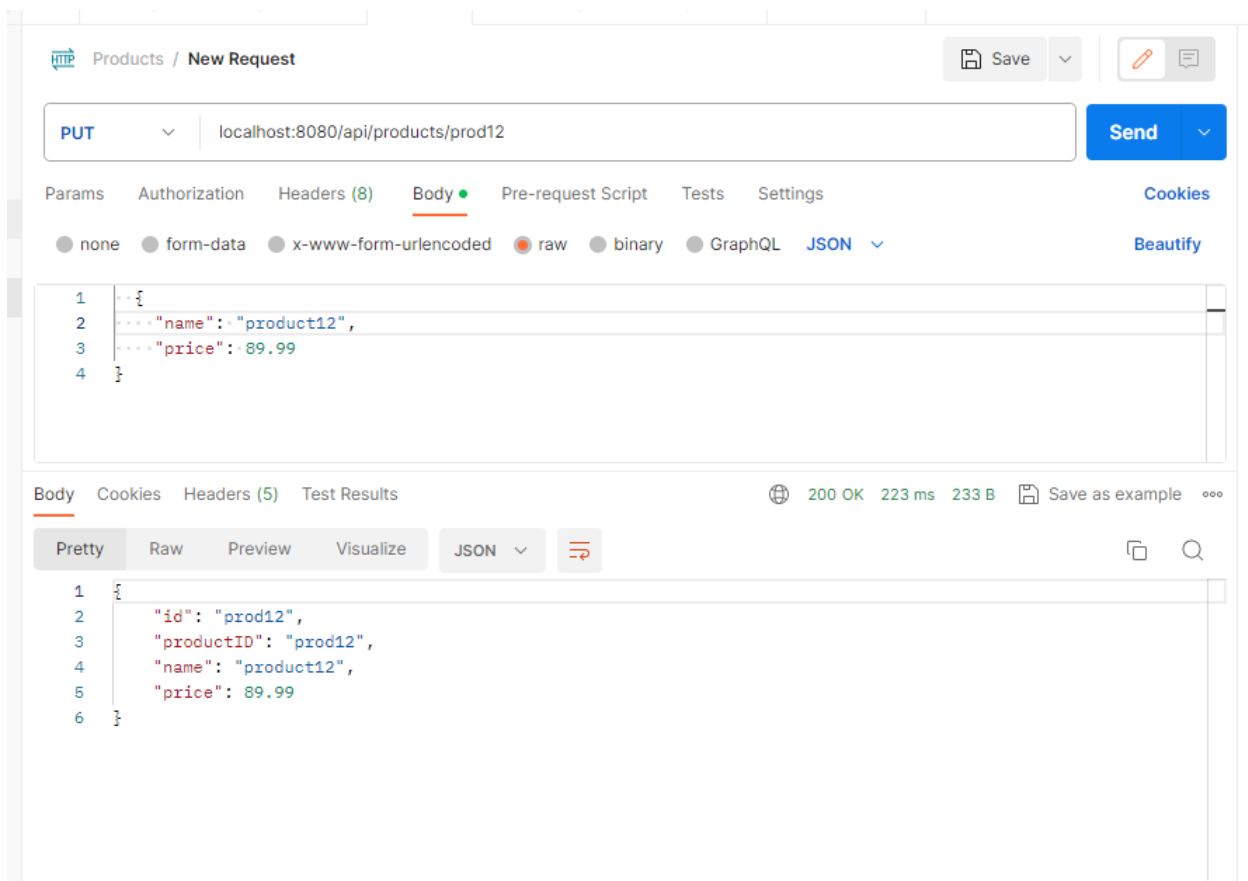


4. Update Products:

- 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.

