

Lesson 03 Demo 02

Create Microservice with MongoDB

Objective: To create a microservice using Spring Boot and MongoDB to perform CRUD operations on a user collection in a MongoDB database

Tool required: Eclipse IDE, MongoDB Atlas, and Postman

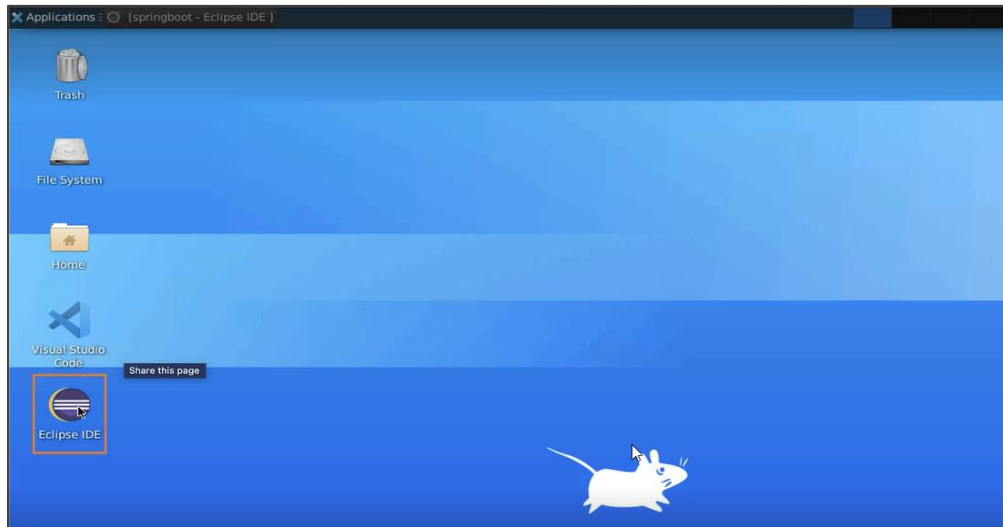
Prerequisites: None

Steps to be followed:

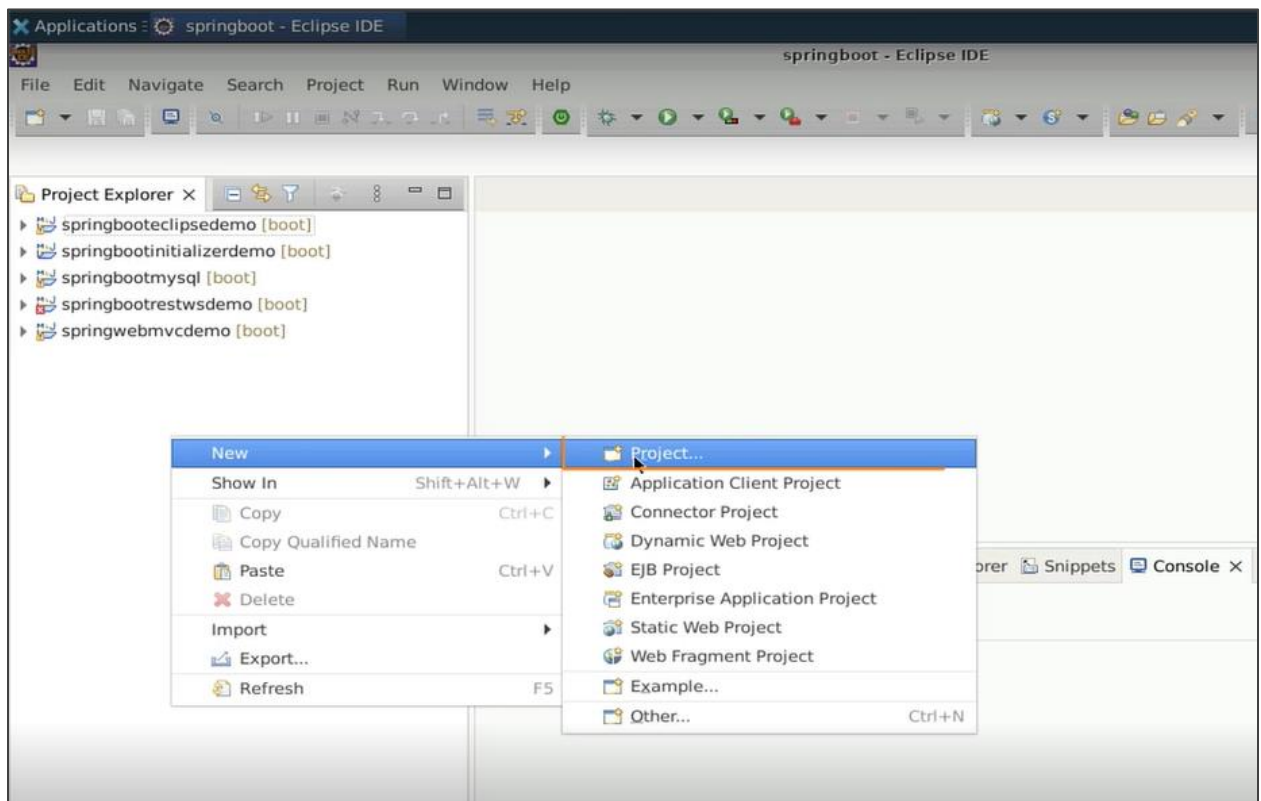
1. Creating a new Spring Starter project
2. Creating the Address Model class
3. Creating the User Model class
4. Configuring the MongoDB database
5. Setting up the database configuration
6. Creating the UserRepository interface
7. Creating the UserController class
8. Creating the Response class
9. Configuring the CRUD methods
10. Running and testing the application

Step 1: Creating a new Spring Starter project

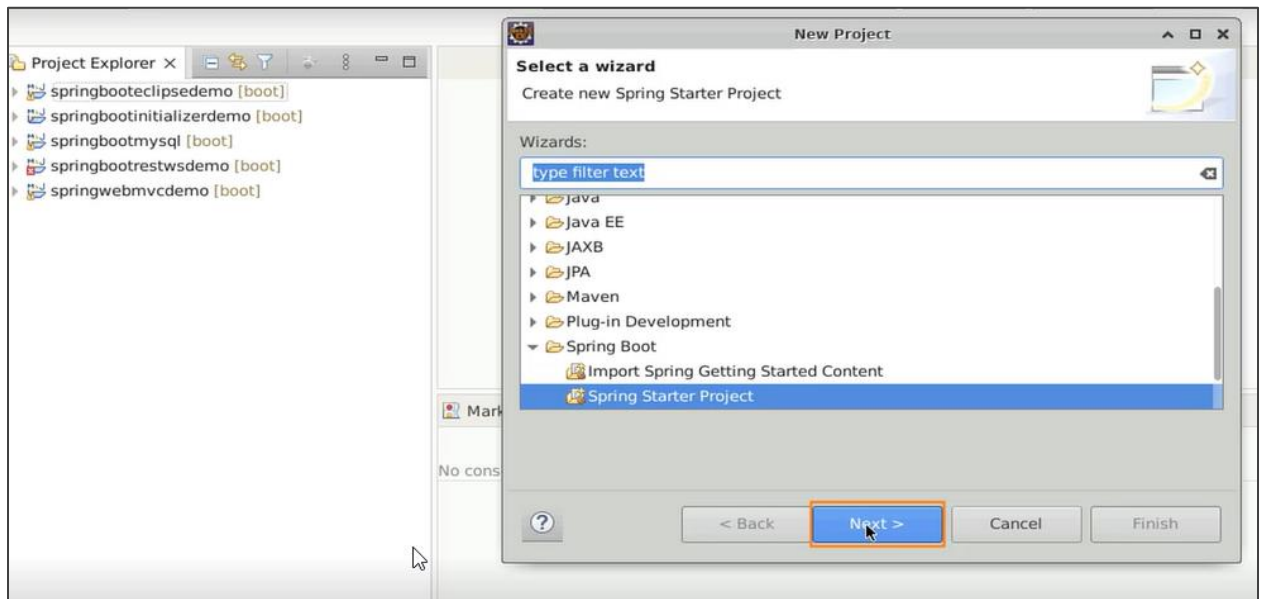
1.1 Open Eclipse IDE



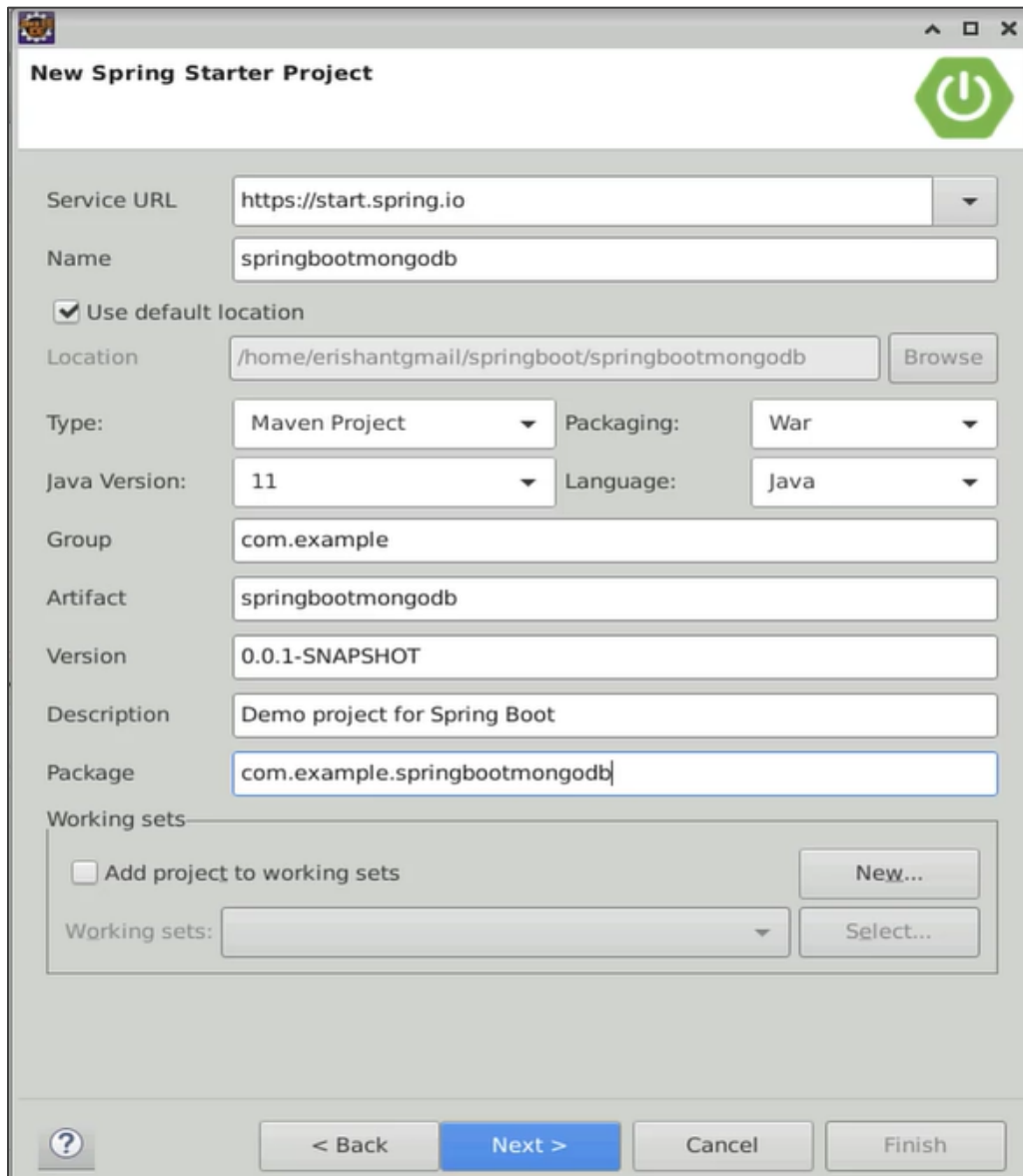
1.2 In the Project Explorer, right-click and select **New > Project**



1.3 Select **Spring Starter Project** and click **Next**



- 1.4 Provide a project name, such as **springbootmongodb**, configure the project with Maven as the build tool, and choose the packaging as **War**



New Spring Starter Project

Service URL:

Name:

☒ Use default location

Location:

Type: Packaging:

Java Version: Language:

Group:

Artifact:

Version:

Description:

Package:

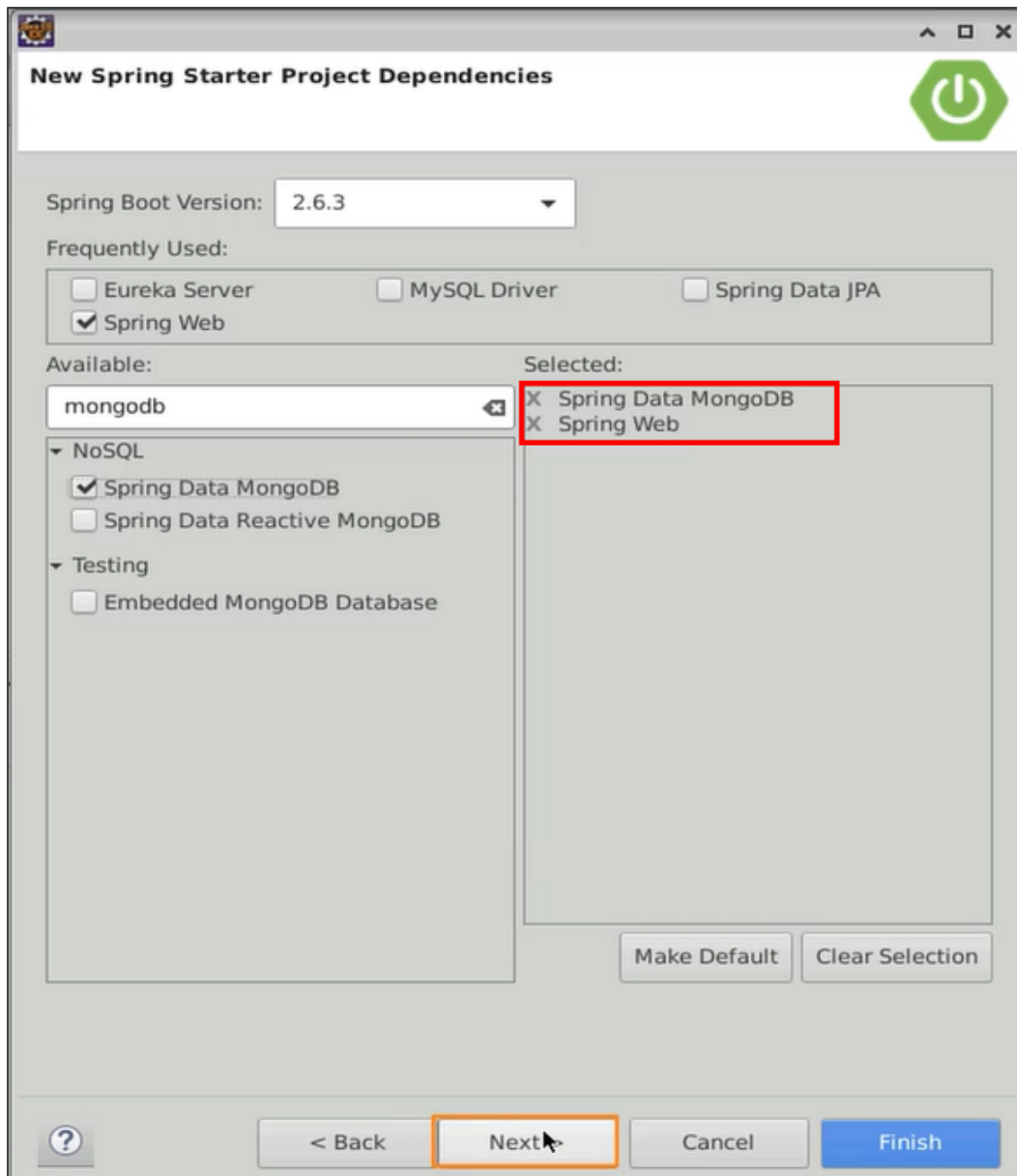
Working sets:

☐ Add project to working sets

Working sets:

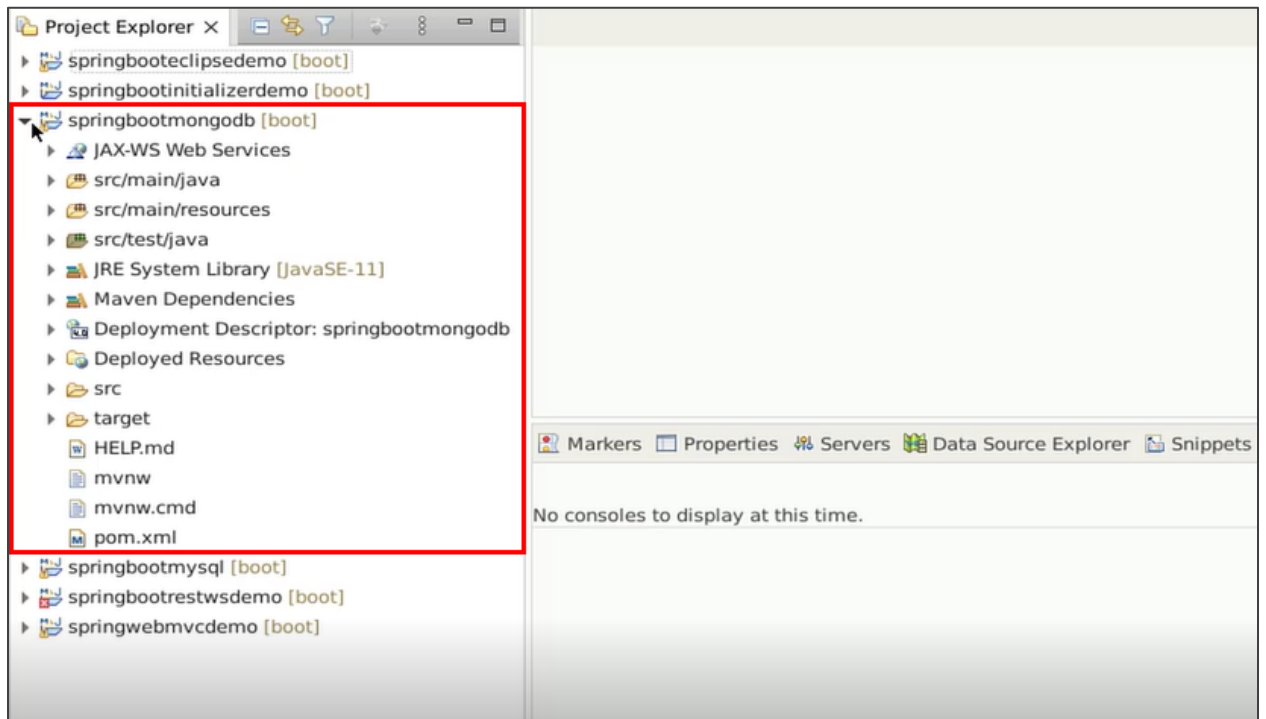
1.5 Click **Next** and add the following dependencies:

- Spring Data MongoDB
- Spring Web



1.6 Click **Finish** to create the project with the specified dependencies

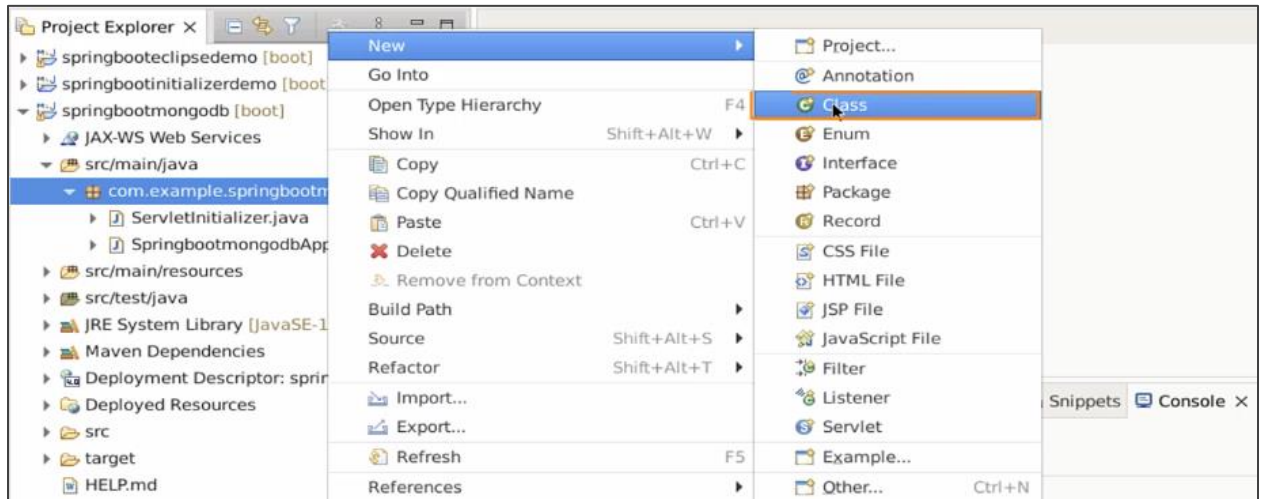




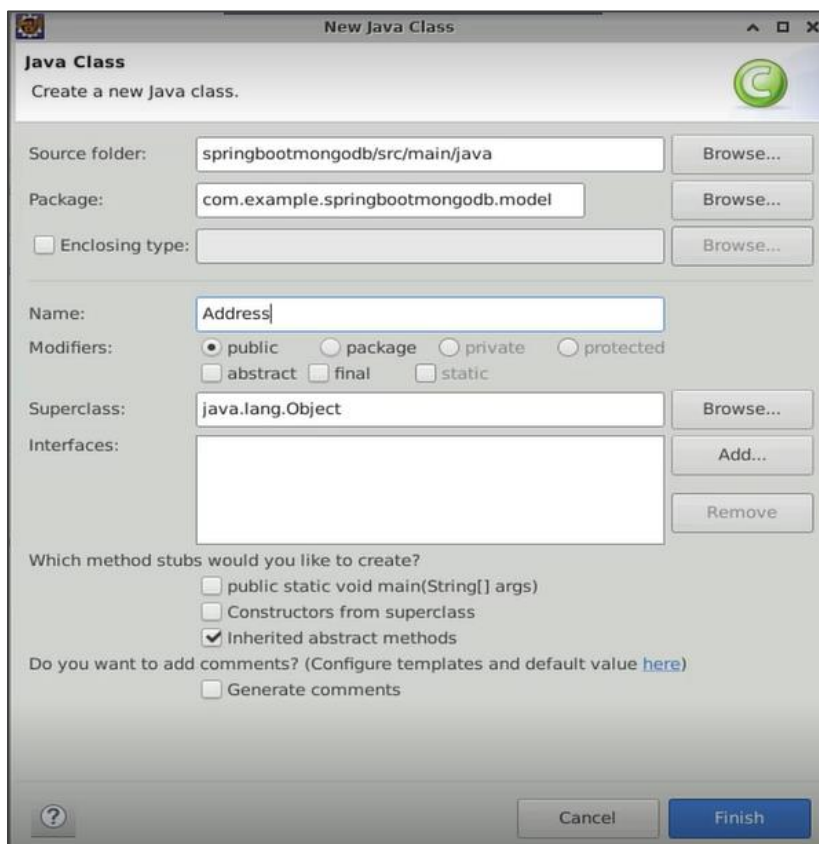
The eclipse will generate the project structure and download the necessary dependencies.

Step 2: Creating the Address Model class

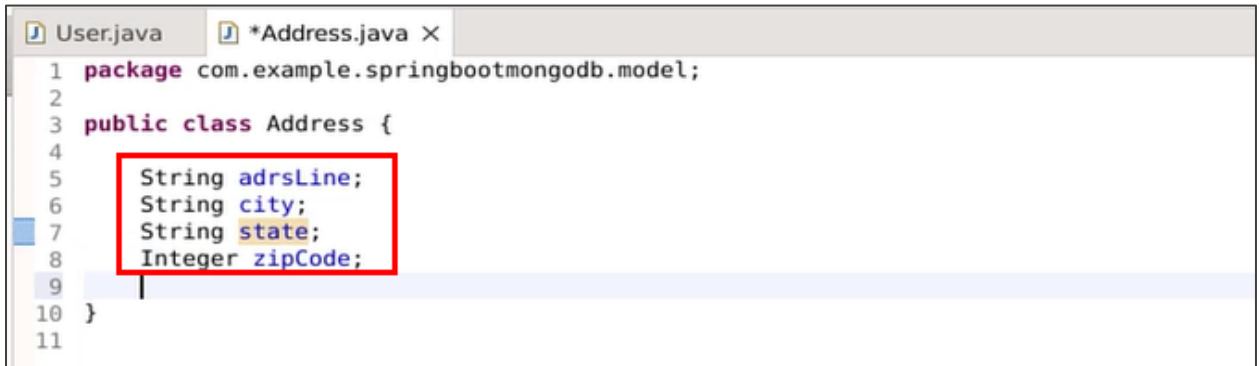
2.1 Right-click on the source package and select **New > Class**



2.2 Name the class **Address** and add **.model** to the package name and click **Finish**



2.3 Add the required attributes for the Address class, such as **adrsLine**, **city**, **state**, and **zipCode**

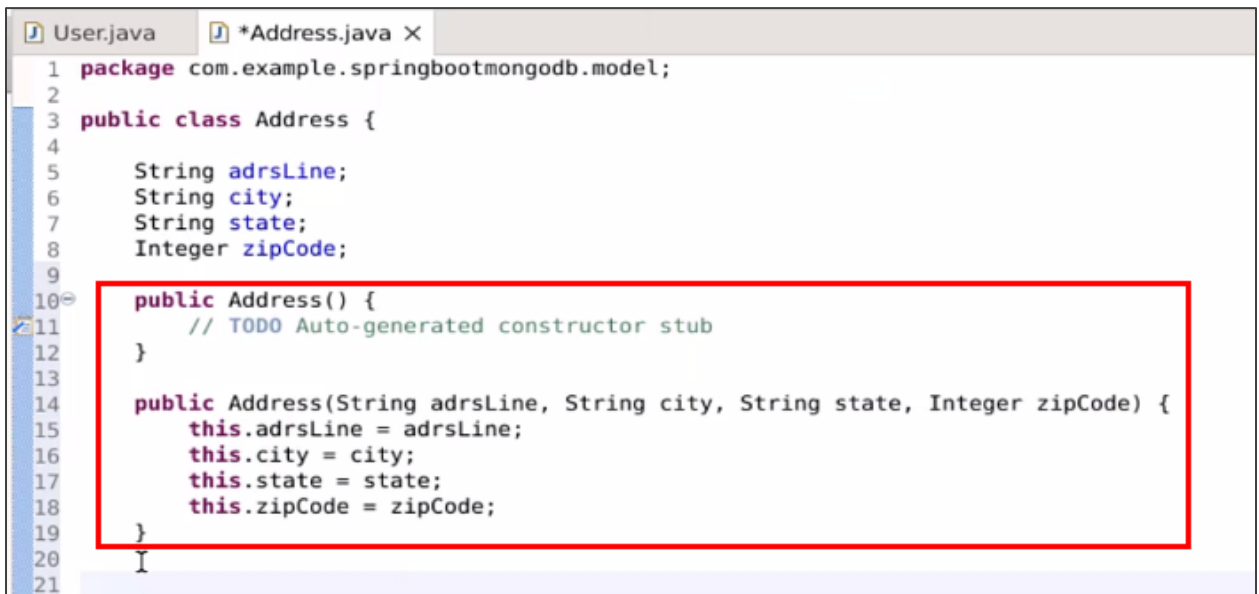


The screenshot shows an IDE with two tabs: 'User.java' and '*Address.java'. The 'Address.java' tab is active, displaying the following code:

```
1 package com.example.springbootmongodb.model;
2
3 public class Address {
4
5     String adrsLine;
6     String city;
7     String state;
8     Integer zipCode;
9
10 }
11
```

A red rectangle highlights the four attributes: `String adrsLine;`, `String city;`, `String state;`, and `Integer zipCode;`.

2.4 Generate the default constructor, parameterized constructor, getters, setters, and a **toString()** method for the Address class



The screenshot shows the same IDE with the 'Address.java' tab active. The code now includes two constructors, which are highlighted by a red rectangle:

```
1 package com.example.springbootmongodb.model;
2
3 public class Address {
4
5     String adrsLine;
6     String city;
7     String state;
8     Integer zipCode;
9
10     public Address() {
11         // TODO Auto-generated constructor stub
12     }
13
14     public Address(String adrsLine, String city, String state, Integer zipCode) {
15         this.adrsLine = adrsLine;
16         this.city = city;
17         this.state = state;
18         this.zipCode = zipCode;
19     }
20
21 }
```

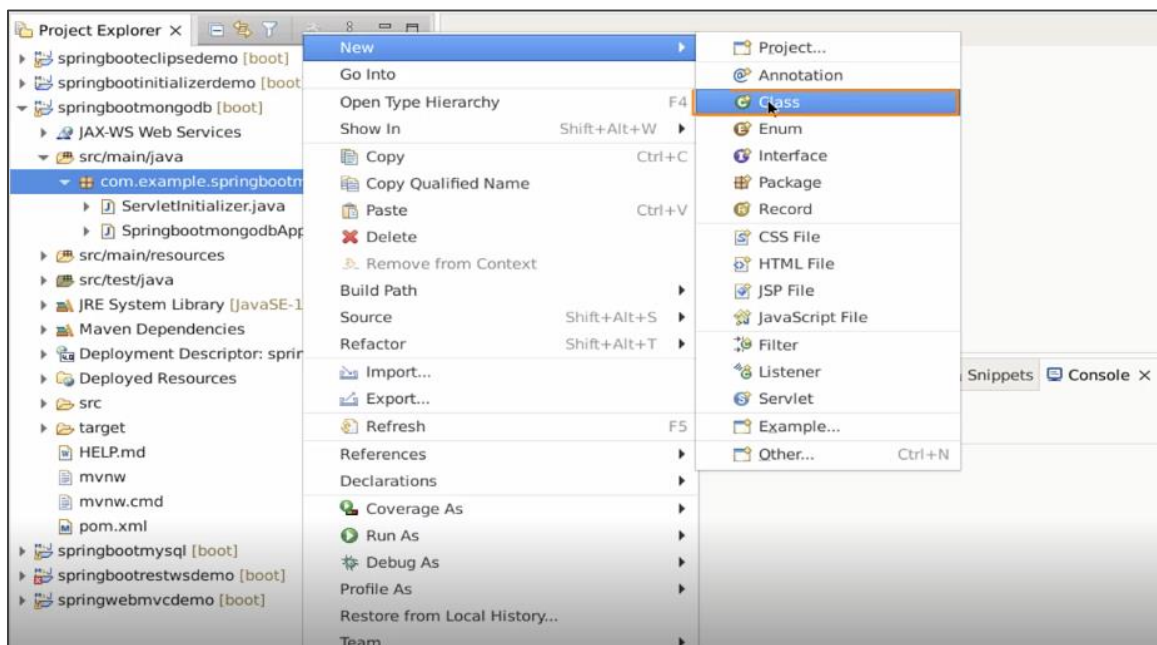
```

User.java  *Address.java X
25 public void setAdrsLine(String adrsLine) {
26     this.adrsLine = adrsLine;
27 }
28
29 public String getCity() {
30     return city;
31 }
32
33 public void setCity(String city) {
34     this.city = city;
35 }
36
37 public String getState() {
38     return state;
39 }
40
41 public void setState(String state) {
42     this.state = state;
43 }
44
45 public Integer getZipCode() {
46     return zipCode;
47 }
48
49 public void setZipCode(Integer zipCode) {
50     this.zipCode = zipCode;
51 }
52
53 @Override
54 public String toString() {
55     return "Address [adrsLine=" + adrsLine + ", city=" + city + ", state=" + state + ", zipCode=" + zipCode + "];"
56 }
57

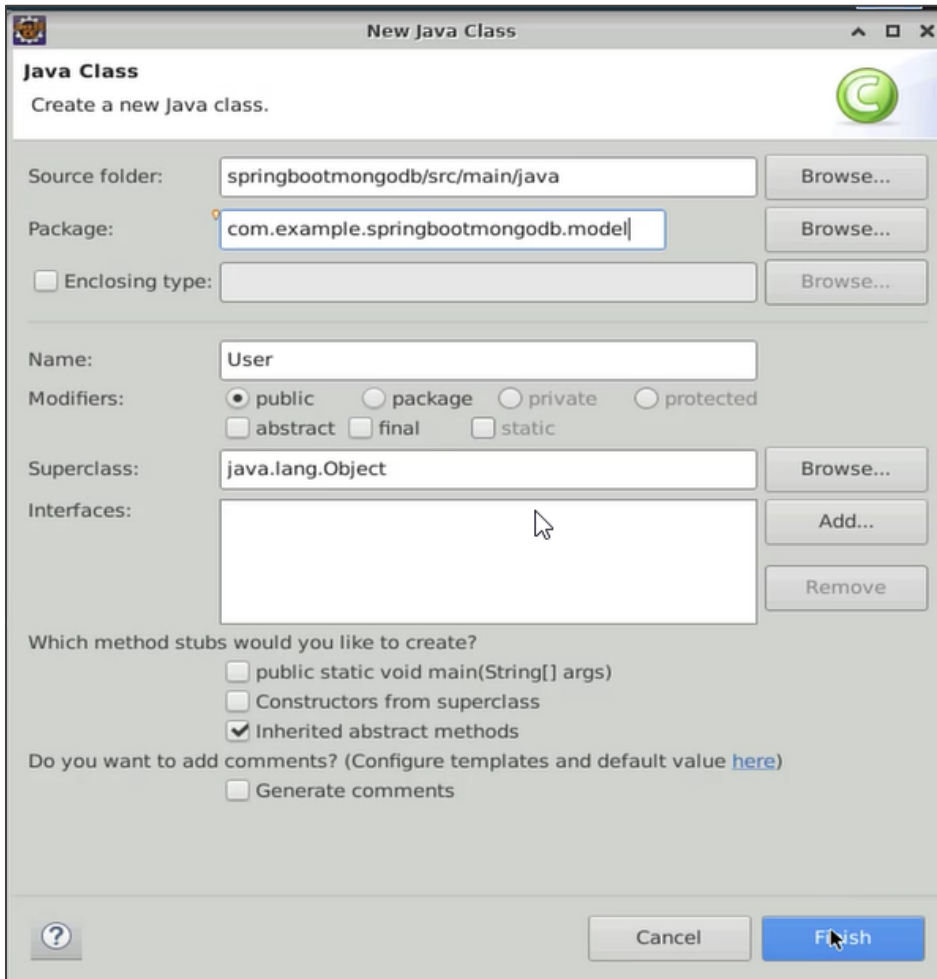
```

Step 3: Creating the User Model class

3.1 Right-click on the source package and select **New > Class**



3.2 Name the class **User** and add **.model** to the package name and click **Finish**



Java Class
Create a new Java class.

Source folder:

Package:

☐ Enclosing type:

Name:

Modifiers: ☒ public ☐ package ☐ private ☐ protected
☐ abstract ☐ final ☐ static

Superclass:

Interfaces:

Which method stubs would you like to create?
☐ public static void main(String[] args)
☐ Constructors from superclass
☒ Inherited abstract methods

Do you want to add comments? (Configure templates and default value [here](#))
☐ Generate comments

3.3 Add fields such as **id** (annotated with **@Id**), **name**, **phone**, **email**, and an **address** object

```
User.java X
1 package com.example.springbootmongodb.model;
2
3 import org.springframework.data.annotation.Id;
4
5 public class User {
6
7     @Id
8     String id;
9
10    String name;
11    String phone;
12    String email;
13
14 }
15
```

```
User.java X Address.java
1 package com.example.springbootmongodb.model;
2
3 import org.springframework.data.annotation.Id;
4
5 public class User {
6
7     @Id
8     String id;
9
10    String name;
11    String phone;
12    String email;
13
14    Address address; // HAS-A Relationship
15
16 }
17
```

3.4 Generate the default constructor, parameterized constructor, getters, setters, and a `toString()` method for the User class

```

1 package com.example.springbootmongodb.model;
2
3 import org.springframework.data.annotation.Id;
4
5 public class User {
6
7     @Id
8     String id;
9
10    String name;
11    String phone;
12    String email;
13
14    Address address; // HAS-A Relationship | 1 to 1
15
16    public User() {
17        // TODO Auto-generated constructor stub
18    }
19
20    public User(String id, String name, String phone, String email, Address address) {
21        this.id = id;
22        this.name = name;
23        this.phone = phone;
24        this.email = email;
25        this.address = address;
26    }
27

```

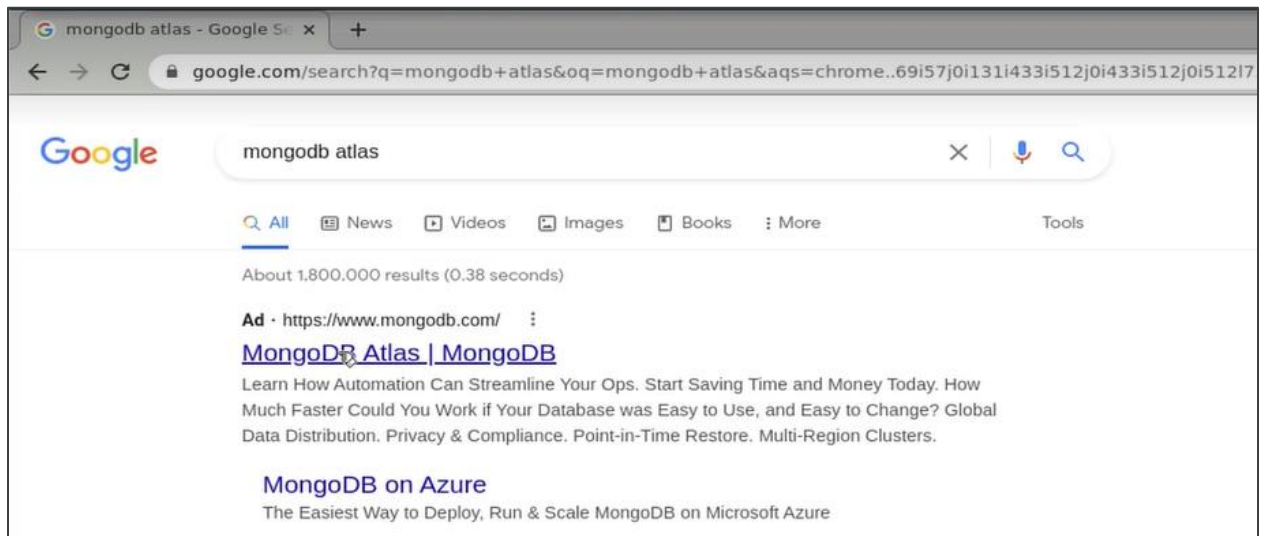
```

40    public void setName(String name) {
41        this.name = name;
42    }
43
44    public String getPhone() {
45        return phone;
46    }
47
48    public void setPhone(String phone) {
49        this.phone = phone;
50    }
51
52    public String getEmail() {
53        return email;
54    }
55
56    public void setEmail(String email) {
57        this.email = email;
58    }
59
60    public Address getAddress() {
61        return address;
62    }
63
64    public void setAddress(Address address) {
65        this.address = address;
66    }
67
68    @Override
69    public String toString() {
70        return "User [id=" + id + ", name=" + name + ", phone=" + phone + ", email=" + email + ", address=" + address
71            + "]";
72    }
73

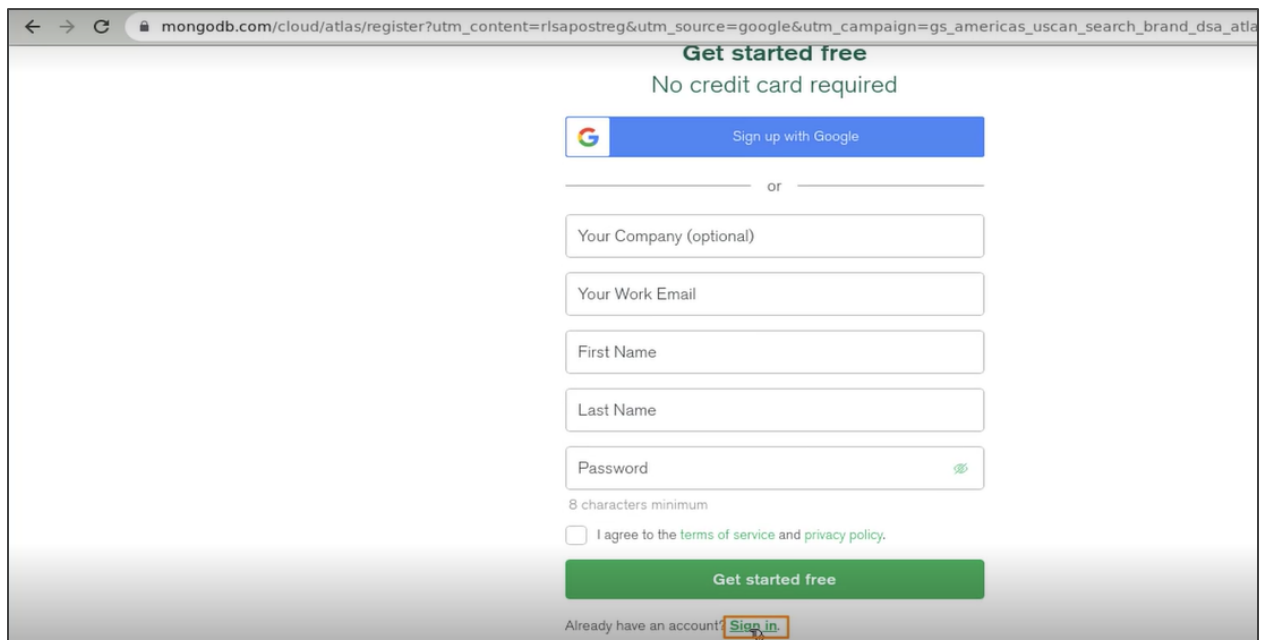
```

Step 4: Configuring the MongoDB database

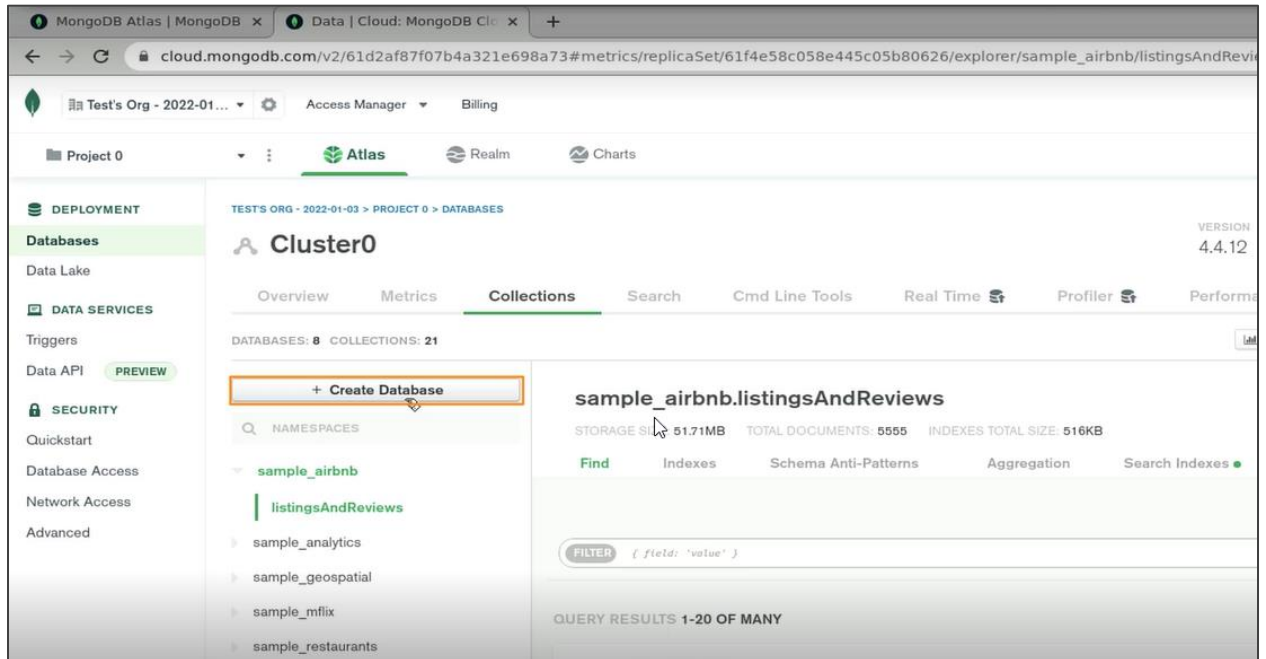
4.1 Open a web browser and navigate to the URL <https://www.mongodb.com>



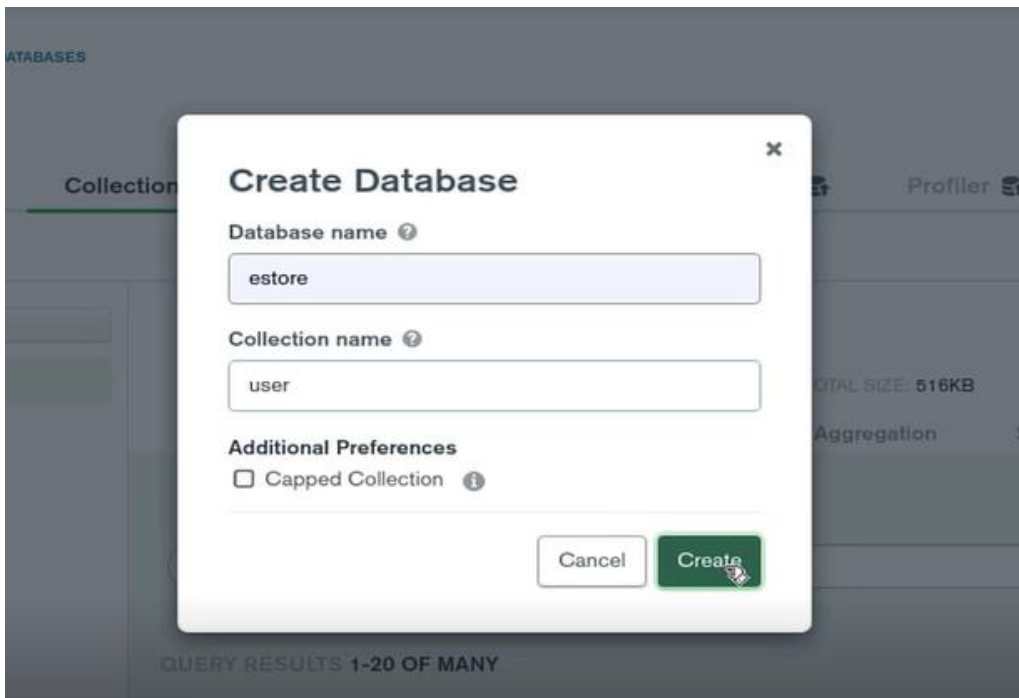
4.2 Register a new account and sign in

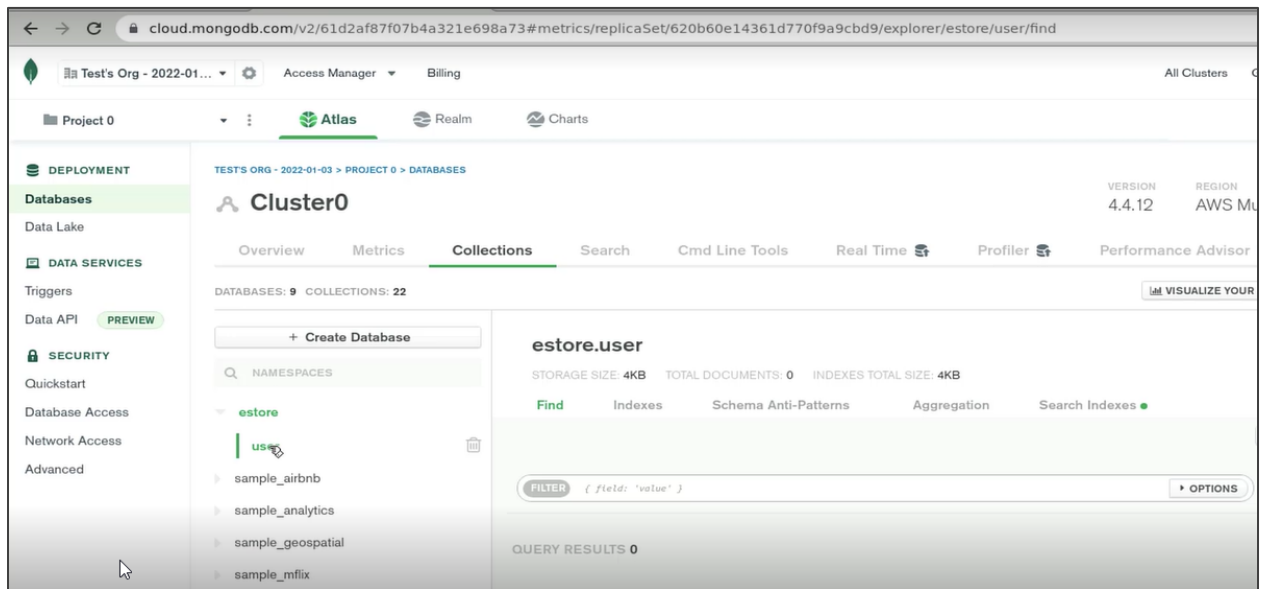


4.3 Create a new database by navigating to the **Collections** tab and clicking on **Create Database**



4.4 Name the database as **estore** and the collection as **user**. Now, click **Create**

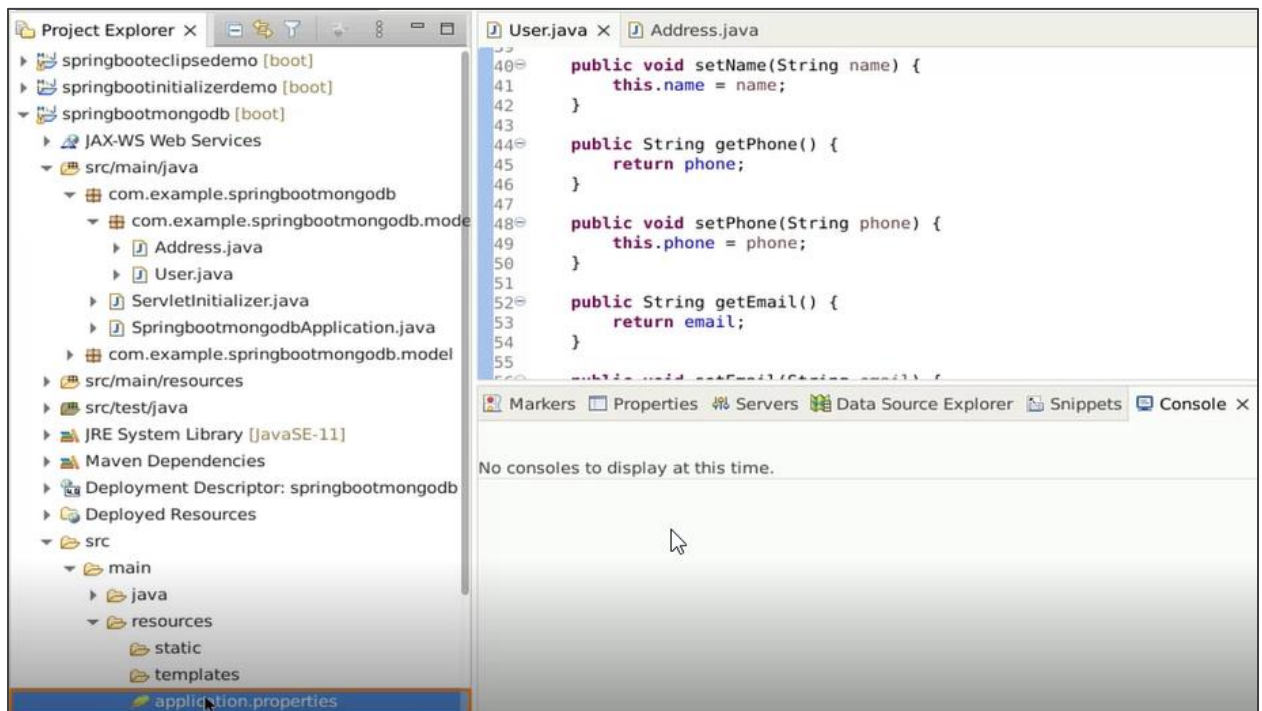




The **estore** database has been created, but currently, there are no records in it.

Step 5: Setting up the database configuration

5.1 Open the **application.properties** file located in the **src/main/resources** directory



5.2 Configure the server port by adding the property **server.port=9090** (choose any port number)

```

1 server.port=9090
2
3
  
```

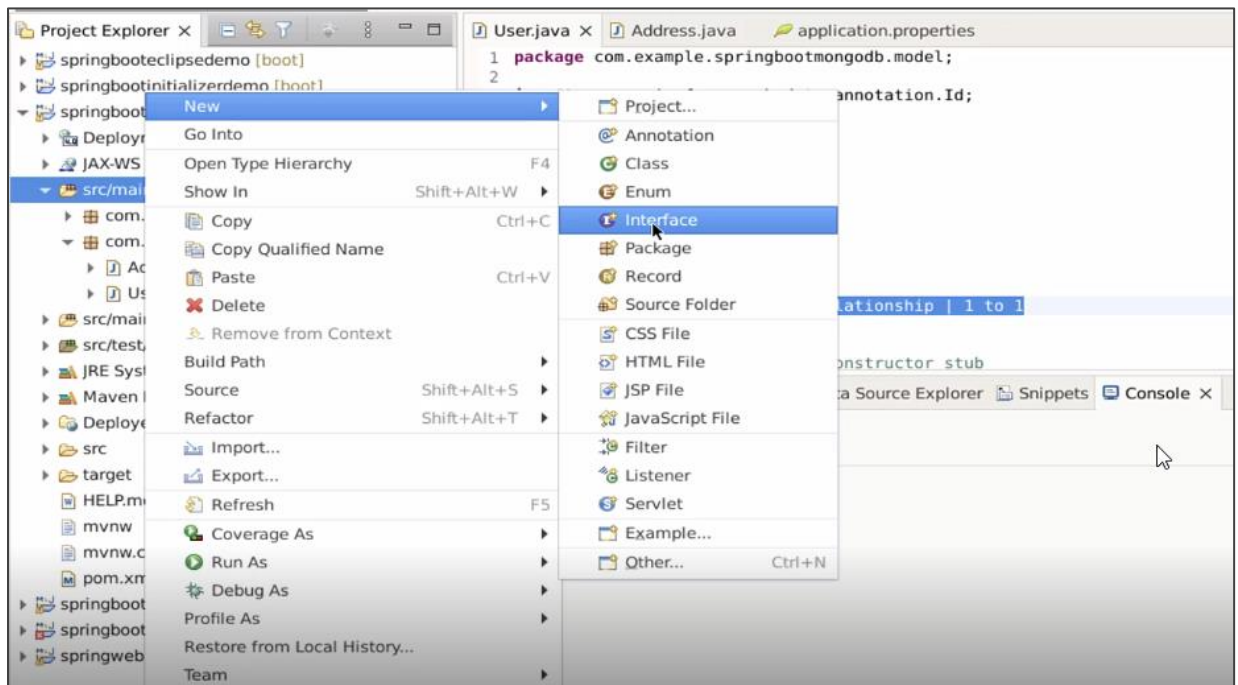
5.3 Specify the database name using **spring.data.mongodb.database** and set the MongoDB URI using **spring.data.mongodb.uri** (obtained from MongoDB Atlas)

```

1 server.port=9090
2 spring.data.mongodb.uri=mongodb+srv://john:john@cluster0.acllp.mongodb.net/myFirstDatabase?retryWrites=true&w=majority
3 spring.data.mongodb.database=estore
4
  
```

Step 6: Creating the UserRepository interface

6.1 Right-click on the source package and select **New > Interface**



6.2 Name the interface **UserRepository** and add **.repository** to the package name and click **Finish**

New Java Interface

Create a new Java interface.

Source folder:

Package:

☐ Enclosing type:

Name:

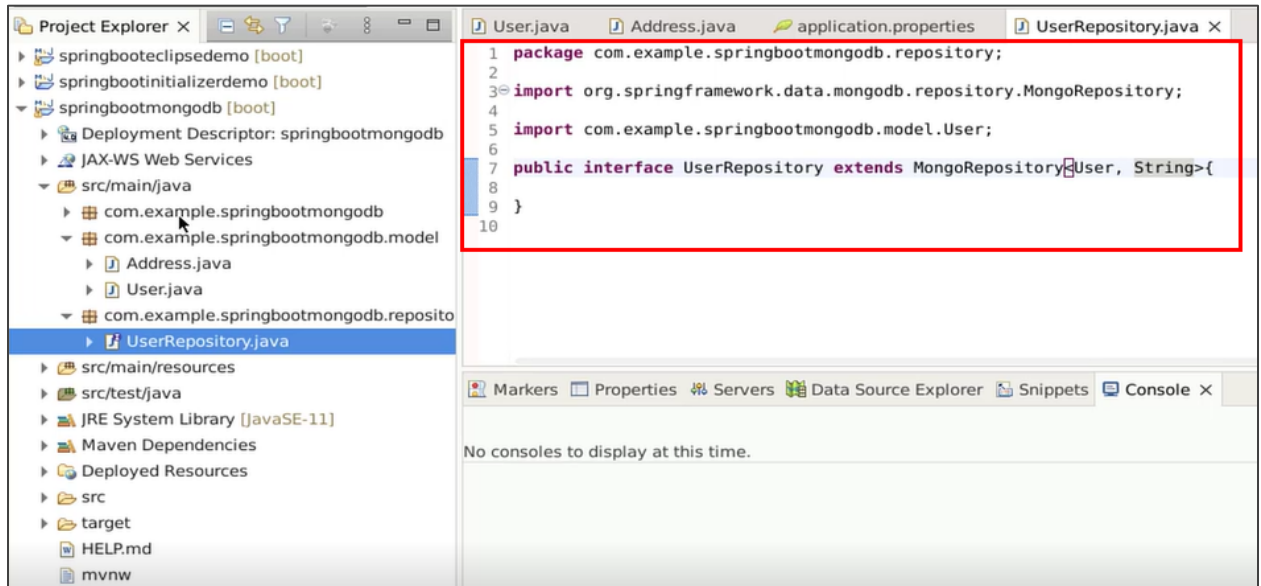
Modifiers: ☒ public ☐ package ☐ private ☐ protected

Extended interfaces:

Do you want to add comments? (Configure templates and default value [here](#))

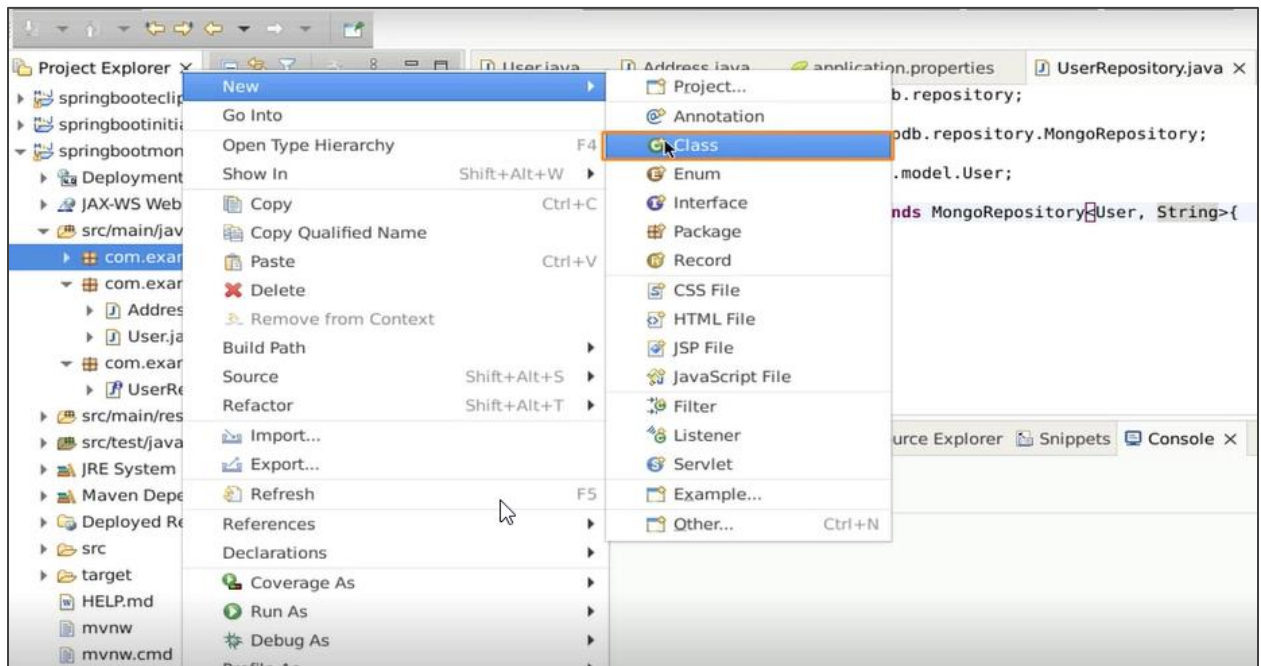
☐ Generate comments

6.3 Extend the **MongoRepository<User, String>** interface to inherit the CRUD operations for the **User** entity

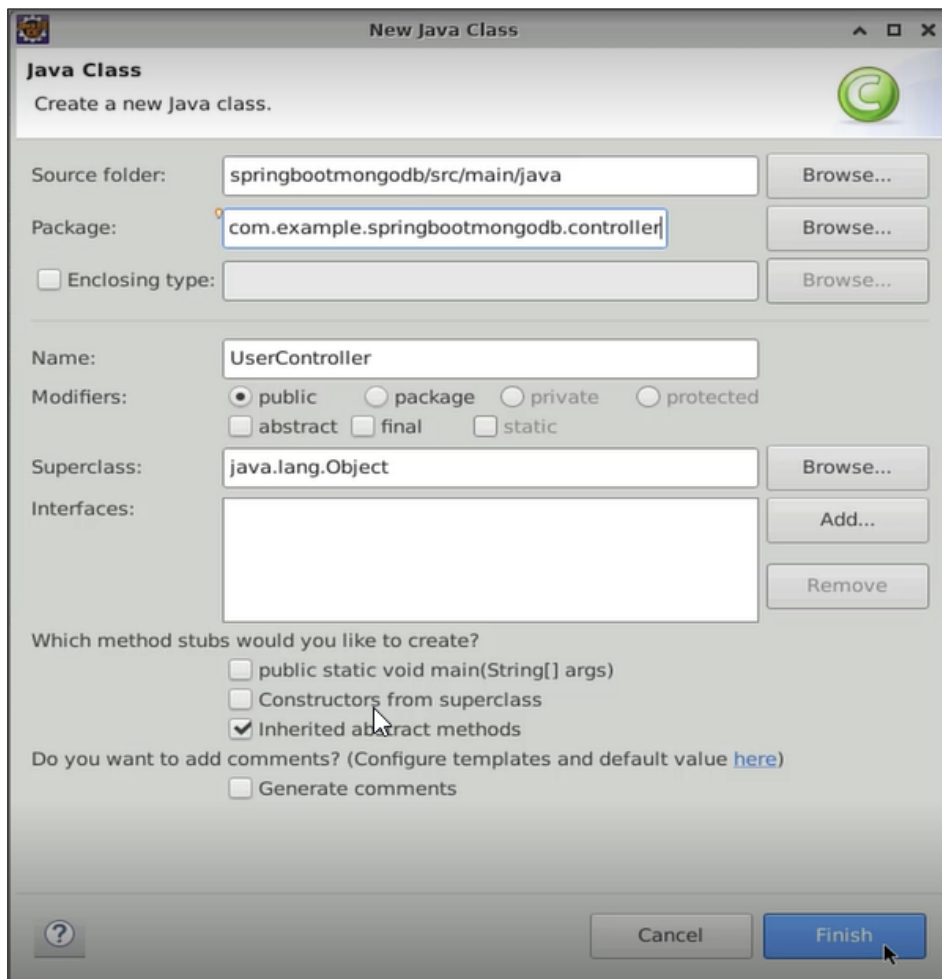


Step 7: Creating the UserController class

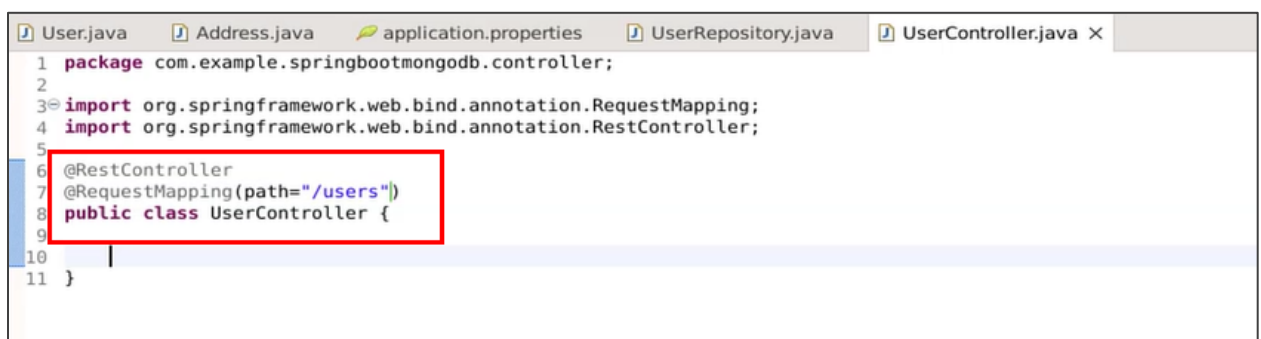
7.1 Right-click on the source package and select **New > Class**



7.2 Name the class **UserController** and add **.controller** to the package name and click **Finish**



7.3 Annotate the class with **@RestController** to indicate it is a controller for handling RESTful requests. Add **@RequestMapping** with the path set to **/users**



7.4 Autowire the **UserRepository** into the controller

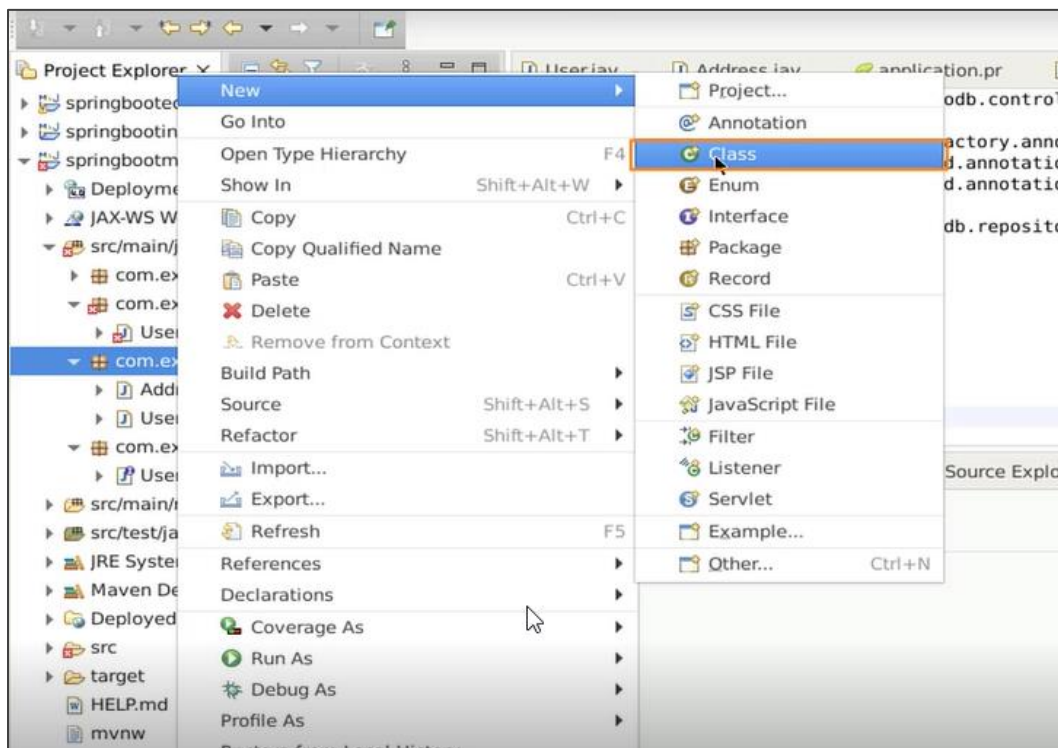
```

1 package com.example.springbootmongodb.controller;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4 import org.springframework.web.bind.annotation.RequestMapping;
5 import org.springframework.web.bind.annotation.RestController;
6
7 import com.example.springbootmongodb.repository.UserRepository;
8
9 @RestController
10 @RequestMapping(path="/users")
11 public class UserController {
12
13     @Autowired
14     UserRepository repository;
15
16 }
17

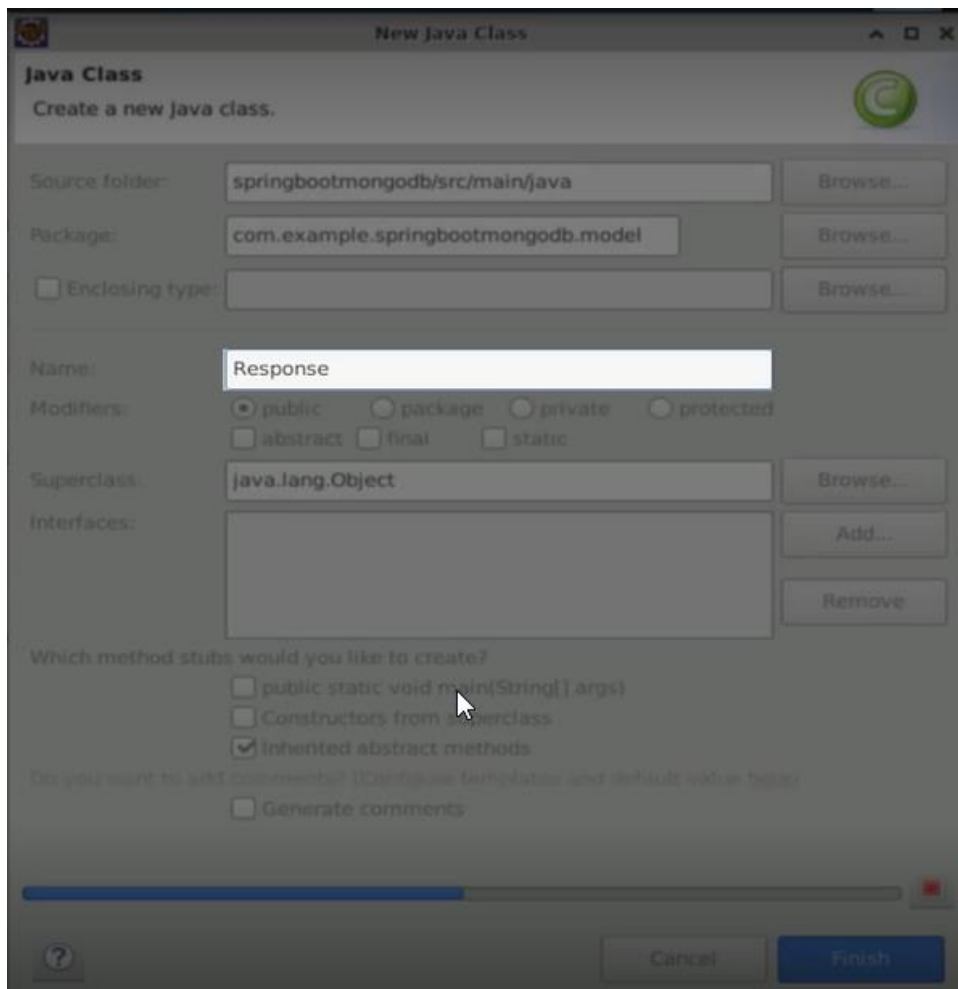
```

Step 8: Creating the Response class

8.1 Right-click on the controller package and select **New > Class**



8.2 Name the class **Response** and click **Finish**



8.3 Define the class with the necessary fields for the response, such as **code**, **message**, and a **List<User>** for users

```
User.java Address.java application.properties UserRepository.java UserController.java *Response.java x
1 package com.example.springbootmongodb.model;
2
3 import java.util.List;
4
5 public class Response {
6
7     Integer code;
8     String message;
9     List<User> users;
10
11     |
12
13 }
14
```

8.4 Generate the default constructor, parameterized constructor, getters, setters, and a `toString()` method for the Response class

```

1 package com.example.springbootmongodb.model;
2
3 import java.util.List;
4
5 public class Response {
6
7     Integer code;
8     String message;
9     List<User> users;
10
11     public Response() {
12         // TODO Auto-generated constructor stub
13     }
14
15     public Response(Integer code, String message, List<User> users) {
16         this.code = code;
17         this.message = message;
18         this.users = users;
19     }
20

```

```

23         this.code = code;
24     }
25
26     public Integer getCode() {
27         return code;
28     }
29
30     public void setCode(Integer code) {
31         this.code = code;
32     }
33
34     public String getMessage() {
35         return message;
36     }
37
38     public void setMessage(String message) {
39         this.message = message;
40     }
41
42     public List<User> getUsers() {
43         return users;
44     }
45
46     public void setUsers(List<User> users) {
47         this.users = users;
48     }
49
50     @Override
51     public String toString() {
52         return "Response [code=" + code + ", message=" + message + ", users=" + users + "]";
53     }

```


Step 9: Configuring the CRUD methods

9.1 In the **UserController.java** class, add a method called **addUser** and annotate it with **@PostMapping** to set the endpoint as **/add**

```
11 import org.springframework.web.bind.annotation.RestController;
12
13 import com.example.springbootmongodb.model.Address;
14 import com.example.springbootmongodb.model.Response;
15 import com.example.springbootmongodb.model.User;
16 import com.example.springbootmongodb.repository.UserRepository;
17
18 @RestController
19 @RequestMapping(path="/users")
20 public class UserController {
21
22     @Autowired
23     UserRepository repository;
24
25     @PostMapping(path="/add")
26     public ResponseEntity<Response> addUser(@RequestParam String name, @RequestParam String phone, @RequestParam String email,
27     @RequestParam String adrLine, @RequestParam String city, @RequestParam String state, @RequestParam Integer zipCode) {
28
```

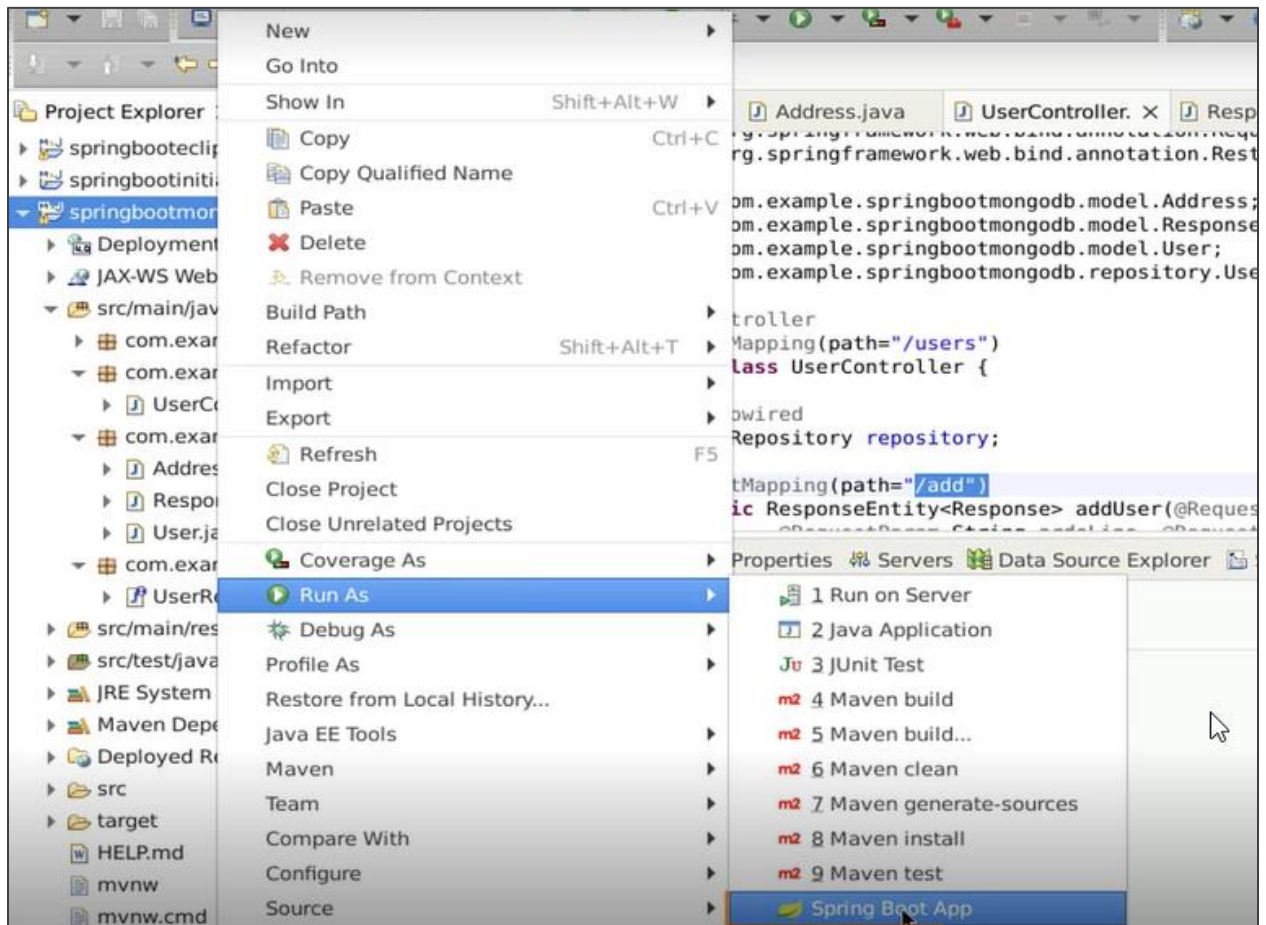
9.2 Implement the logic to save the user details in the database and return a status code of **101**. If any exceptions occur, catch them using a try-catch block and return a status code of **301**

```
11 import org.springframework.web.bind.annotation.RestController;
12
13 import com.example.springbootmongodb.model.Address;
14 import com.example.springbootmongodb.model.Response;
15 import com.example.springbootmongodb.model.User;
16 import com.example.springbootmongodb.repository.UserRepository;
17
18 @RestController
19 @RequestMapping(path="/users")
20 public class UserController {
21
22     @Autowired
23     UserRepository repository;
24
25     @PostMapping(path="/add")
26     public ResponseEntity<Response> addUser(@RequestParam String name, @RequestParam String phone, @RequestParam String email,
27     @RequestParam String adrLine, @RequestParam String city, @RequestParam String state, @RequestParam Integer zipCode) {
28
29         try {
30
31             Address address = new Address(adrLine, city, state, zipCode);
32             User user = new User(null, name, phone, email, address);
33
34             repository.save(user);
35
36             Response response = new Response(101, " User "+name+" Saved Successfully at " + new Date());
37             return new ResponseEntity<Response>(response, HttpStatus.INTERNAL_SERVER_ERROR);
38
39         } catch (Exception e) {
40             Response response = new Response(301, "OOPS!! Something Went Wrong: "+e.getMessage());
41             return new ResponseEntity<Response>(response, HttpStatus.INTERNAL_SERVER_ERROR);
42         }
43     }
44 }
```

Similarly, you can implement other methods for update, delete, and read operations for the user collection.

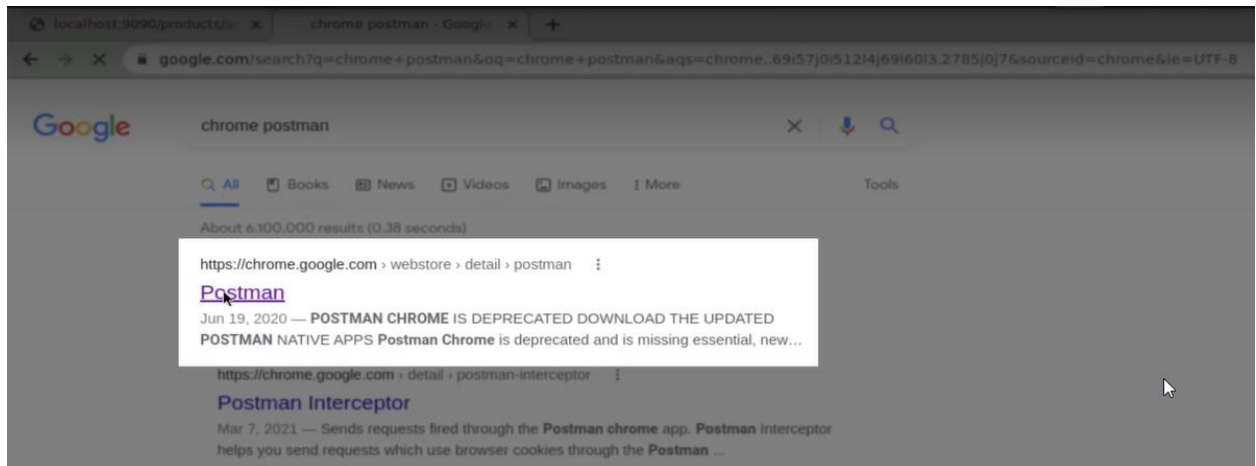
Step 10: Running and testing the application

10.1 Right-click on the project and select **Run As > Spring Boot App** to start the application

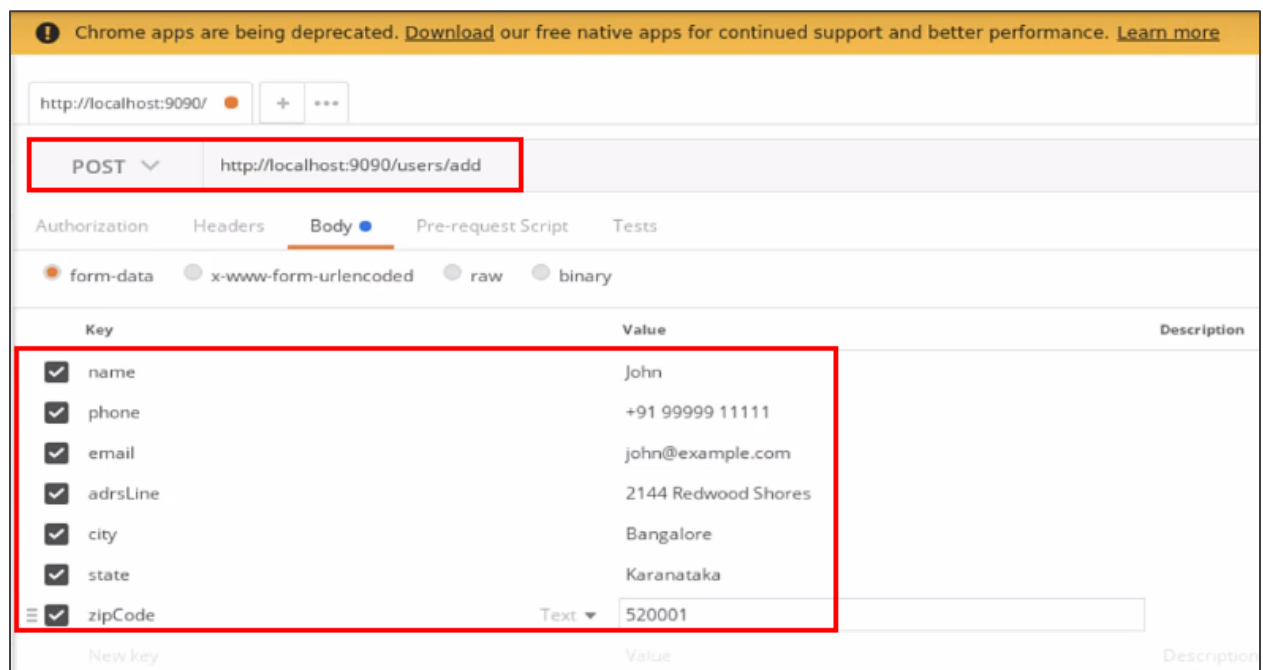


Spring Boot will automatically start the embedded Tomcat Server and deploy the application at **localhost:9090**

10.2 To test the **addUser** method, open Postman in a web browser



10.3 On the main page, send a POST request to the URL **http://localhost:9090/users/add**, which forwards the request to the **addUser()** controller method. Define the data for the user collection using key-value pairs under the **Body** section



10.4 Click **Send**

http://localhost:9090/ + ... No Environment

POST http://localhost:9090/users/add Params **Send**

Authorization Headers **Body** Pre-request Script Tests

form-data x-www-form-urlencoded raw binary

Key	Value	Description
<input checked="" type="checkbox"/> name	John	
<input checked="" type="checkbox"/> phone	+91 99999 11111	
<input checked="" type="checkbox"/> email	john@example.com	
<input checked="" type="checkbox"/> adrsLine	2144 Redwood Shores	
<input checked="" type="checkbox"/> city	Bangalore	
<input checked="" type="checkbox"/> state	Karnataka	
<input checked="" type="checkbox"/> zipCode	Text 520001	

New key Value Description

http://localhost:9090/ + ... No Environment

POST http://localhost:9090/users/add Params **Send**

Authorization Headers **Body** Pre-request Script Tests

form-data x-www-form-urlencoded raw binary

Key	Value	Description
<input checked="" type="checkbox"/> name	John	
<input checked="" type="checkbox"/> phone	+91 99999 11111	
<input checked="" type="checkbox"/> email	john@example.com	
<input checked="" type="checkbox"/> adrsLine	2144 Redwood Shores	
<input checked="" type="checkbox"/> city	Bangalore	
<input checked="" type="checkbox"/> state	Karnataka	
<input checked="" type="checkbox"/> zipCode	520001	

New key Value Description

Body Cookies Headers (4) Test Results Status: 500 Internal Server Error

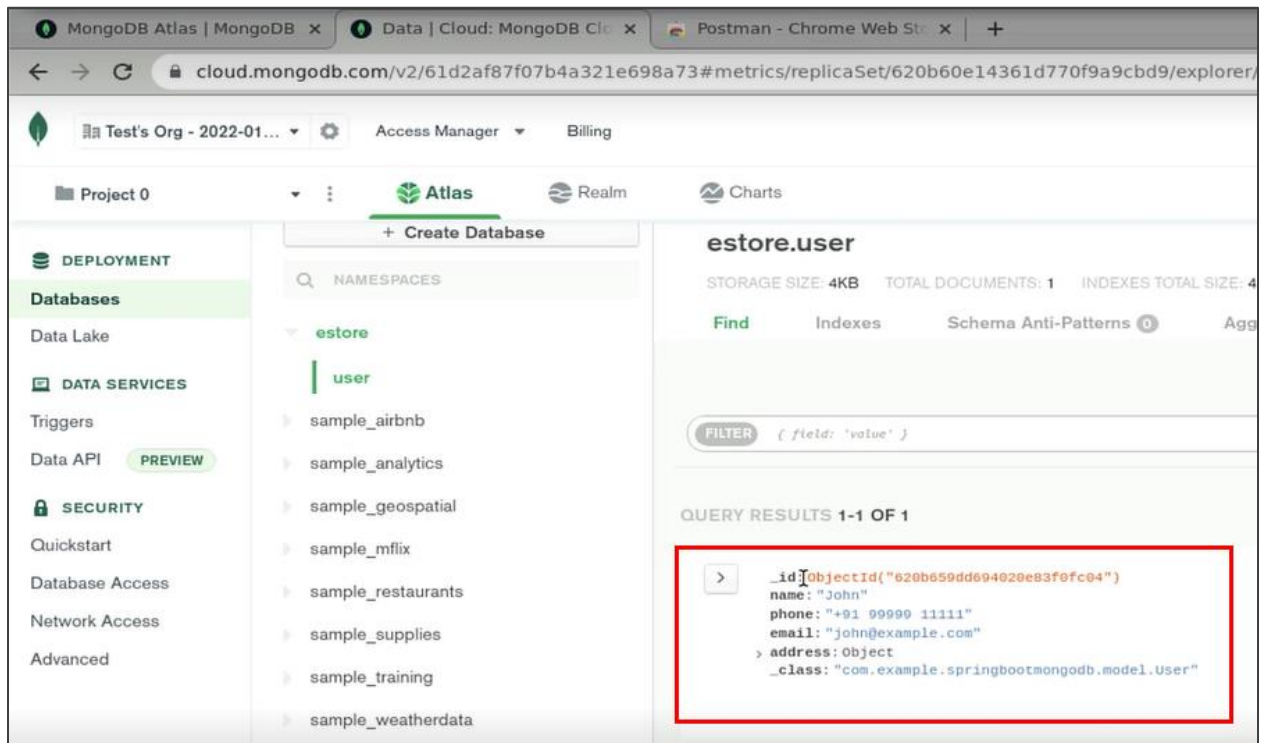
Pretty Raw Preview JSON

```

1 {
2   "code": 101,
3   "message": "User John Saved Successfully at Tue Feb 15 08:34:39 UTC 2022",
4   "users": null
5 }
```

You will see the JSON output with a message stating **User John Saved Successfully** and a status code of **101**.

10.5 Return to the user collection in the MongoDB Atlas and refresh the page



You can see that the user **John** has been added to the user collection as specified in Postman.

Similarly, you can implement other methods to read, update, and delete users in the **UserController.java** class.