



CST-339 Activity 5 Guide

Contents

Part 1: Creating Data Services using Spring Data MongoDB	1
Part 2: Adding New Queries in the MongoDB Repository	3
Appendix A: Setting Up a MongoDB Atlas Project	6

Part 1: Creating Data Services using Spring Data MongoDB

Overview

Goal and Directions:

In this activity, you will code a data service using the Repository design pattern to persist data to a non-relational MongoDB database using Spring Data MongoDB. We will use the standard CRUD Repository that was built in the previous activity and port this over to use MongoDB. The online MongoDB Atlas service will be used for the MongoDB database.

Execution

Execute this assignment according to the following guidelines:

1. Setup a MongoDB Atlas project per the instructions in Appendix A.
2. Make a copy of the *topic4-2* project by right-clicking on the project, selecting the Copy menu option, then right-clicking in the Workspace, and then selecting the Paste menu option. Name the project *topic5-1*. Rename the Spring application file from *Topic42Application.java* to *Topic51Application.java*. Fixup the *artifactId* and *name* tags in the POM file (pom.xml) so the value is set to *topic5-1*. Run the project to ensure that everything is working properly.
3. Make the following changes to the project:
 - a. Remove the *spring-boot-starter-data-jdbc* and the *mysql-connector-java* dependencies from the POM file.
 - b. Add the *spring-boot-starter-data-mongodb* Maven dependency to the POM file.

```
<!-- For Spring Data MongoDB -->
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-mongodb</artifactId>
</dependency>
```

- c. Update *application.properties* file to use the MongoDB Atlas connection configuration:
 - i. Remove existing MySQL database properties.
 - ii. Add a *spring.data.mongodb.uri* property set to the value of `mongodb+srv://<USERNAME>:<PASSWORD>@<CIUSTER-NAME>-`



<INSTANCE-ID>/<DATABASE-NAME>?retryWrites=true (you can get this in your MongoDB Atlas settings).

- iii. Add a *spring.data.mongodb.database* property and set this to the database name (i.e., cst339).

```
# MongoDB Atlas Database
spring.data.mongodb.uri=mongodb+srv://[redacted]@cluster0-ihiam.mongodb.net/test?retryWrites=true&w=majority
spring.data.mongodb.database=cst339
```

- d. Update *OrderEntity.java* file as follows:
 - i. Remove all current annotations and imports.
 - ii. Change the *id* property from a Long to a String.
 - iii. Add the *@Document(collection="orders")* annotation at the class level.
 - iv. Add the *@Id* annotation to the *id* property.
 - v. Add the *@Indexed(unique=true)* annotation to the *orderNo* and *productName* properties.

```
import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.index.Indexed;
import org.springframework.data.mongodb.core.mapping.Document;

@Document(collection="orders")
public class OrderEntity
{
    @Id
    String id;

    @Indexed(unique=true)
    String orderNo;

    @Indexed(unique=true)
    String productName;

    float price;
    int quantity;
}
```

- e. Update *OrderModel.java* file as follows:
 - i. Change the *id* property from a Long to a String.
- f. Remove the *OrderRowMapper.java* file and package from the project.
- g. Change the Repository from a *CrudRepository* type to *MongoRepository* and change the ID key field from a Long to a String.

```
public interface OrdersRepository extends MongoRepository<OrderEntity, String>
{
}
```

- 4. Run the application. Test the Repository *findAll()* and *save()* methods. Open a browser and go to *localhost:8080/login/*. Submit the form to display the Orders page. Verify that the Orders are displayed correctly from the database. Take a screenshot of the Orders page. Also, test the REST API */getjson* and */getxml* API's. take a screenshot of the JSON and XML responses.



5. Update the POM file to name the output JAR file *cst339activity*. Run a Maven build and test the JAR file out from a terminal window. Reference the steps as needed from the Activity 1 Guide.
6. For extra practice (not graded), complete the implementation of the *findById()*, *update()* and *delete()* methods of the *OrdersDataService* and test them within your application.

Deliverables:

The following needs to be submitted as this part of the activity:

- a. Screenshot of the Orders page.
- b. Screenshot of the JSON and XML REST API responses.

Part 2: Adding new queries in the MongoDB Repository

Overview

Goal and Directions:

In this activity, you will add a new finder method to the MongoDB Repository and add a new REST API with proper return response codes.

Execution

Execute this assignment according to the following guidelines:

1. Change the *findById(int id)* method in the *DataAccessInterface* class to *findById(String id)* since MongoDB uses string types for its identifier field.
2. Add a *public getOrderById(String id)* declaration to the *OrdersRepository* class.

```
public interface OrdersRepository extends MongoRepository<OrderEntity, String>
{
    OrderEntity getOrderById(String id);
}
```

3. Implement the *findById()* method in the *OrdersDataService* class by calling the *getOrderById()* method from the *ordersRepository* variable.

```
/**
 * CRUD: finder to return a single entity
 */
public OrderEntity findById(String id)
{
    return ordersRepository.getOrderById(id);
}
```



4. Update the *OrdersBusinessServiceInterface* class by adding a new method *public OrderModel getOrderById(String id)* declaration.

```
public interface OrdersBusinessServiceInterface
{
    public void test();
    public List<OrderModel> getOrders();
    public OrderModel getOrderById(String id);
    public void init();
    public void destroy();
}
```

5. Update the *AnotherOrdersBusinessService* class by implementing the *getOrderById()* method. The implementation can simply return null.
6. Update the *OrdersBusinessService* class by implementing the *getOrderById()* method. The implementation should call the *findById()* method from the *service* variable. Create an instance of an *OrderModel* from the *OrderEntity* returned from the service call. Return the instance of the *OrderModel*.

```
@Override
public OrderModel getOrderById(String id)
{
    // Get the Entity Order
    OrderEntity orderEntity = service.findById(id);

    // Convert Entity Order to Domain Order
    return new OrderModel(orderEntity.getId(), orderEntity.getOrderNo(), orderEntity.getProductName(), orderEntity.getPrice(), orderEntity.getQuantity());
}
```

7. Update the *OrdersRestService* class by adding a new API named *getOrder()* that is mapped to the */getorder/{id}* path and will be used to get an order (i.e., *OrderModel*) given an *id* as a parameter in the URI. The API should be mapped to a GET Request. The *OrderModel* should be obtained by calling the *findById()* method of *service* variable. The API should return an instance of a *ResponseEntity*. Set the *ResponseEntity* status to *HttpStatus.NOT_FOUND* if the return from the service call is null. If the service call returns a non-null value, set the *ResponseEntity* status to *HttpStatus.OK* and set the return entity to the value returned from the service call. Put a try catch block around all the logic and set the *ResponseEntity* status to *HttpStatus.INTERNAL_SERVER_ERROR* in the catch block.

```
@GetMapping(path="/getorder/{id}")
public ResponseEntity<?> getOrder(@PathVariable("id") String id)
{
    try
    {
        OrderModel order = service.getOrderById(id);
        if(order == null)
            return new ResponseEntity<>(HttpStatus.NOT_FOUND);
        else
            return new ResponseEntity<>(order, HttpStatus.OK);
    }
    catch(Exception e)
    {
        return new ResponseEntity<>(HttpStatus.INTERNAL_SERVER_ERROR);
    }
}
```



8. Run the application. Open a browser and go to *localhost:8080/service/getorder/[ID]* where ID is a valid order ID from the MongoDB database. Take a screenshot of the formatted JSON displayed in the browser for a known ID and an invalid ID.

Deliverables:

The following needs to be submitted as this part of the activity:

- a. Screenshot of the JSON REST API response with a known ID.
- b. Screenshot of the JSON REST API response with an invalid ID.

Research Questions

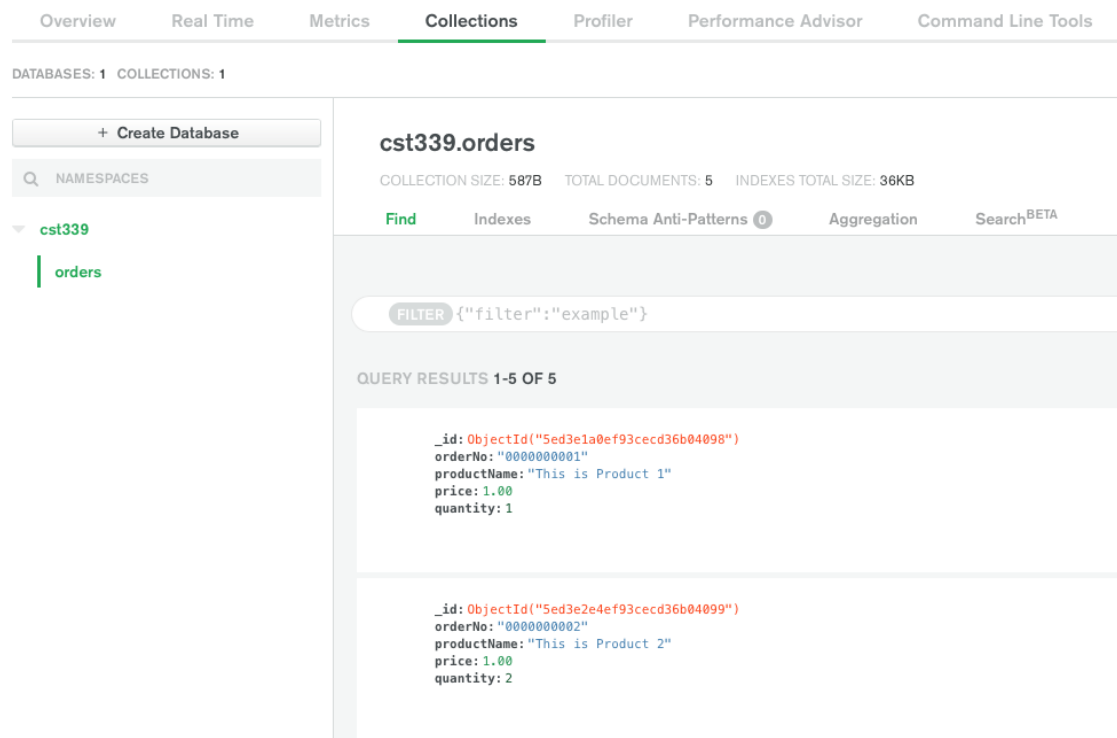
1. Research Questions: For traditional ground students, in a Microsoft Word document, answer the following questions:
 - a. Compare and contrast the design of schema in a relational database and a non-relational database. How do they differ and what impact(s) does migrating from a relational database to a non-relational database have on your application design?
 - b. How does Spring Data support transaction management and the ACID principle?

Final Activity Submission

1. In a Microsoft Word document, complete the following for the Activity Report:
 - a. Cover Sheet with the name of this assignment, date, and your name.
 - b. Section with a title that contains all the screenshots for each part of the activity.
 - c. Section with a title that contains the answers to the research questions (traditional ground students only).
2. Submit the Activity Report as directed by your instructor.

Appendix A: Setting up a MongoDB Atlas Project

1. Go to <https://www.mongodb.com> and create an account.
2. Create a new Project named *CST-339*. Click the *Next* button. Click the *Create Project* button.
3. Create a Cluster by selecting *Clusters* under the left pane. Click the *Build a Cluster* button. Select the free Shared Cluster option. Click the *Create a Cluster* button. Select a region in the US on AWS. Click the *Create Cluster* button.
4. Add a Database and Collection:
 - a. Click the *Collections* button.
 - b. Click the *Add My Own Data* button.
 - c. Create a Database named *cst339*.
 - d. Create a Collection named *orders*.
 - e. Click the *Create* button.
5. Add some Test Data:
 - a. Click the *Insert Document* button.
 - b. Populate the Document with the following properties for an Order and populate some test data:
 - i. String *orderNo*
 - ii. String *productName*
 - iii. Decimal128 *price*
 - iv. Int32 *quantity*
 - v. Click the *Clone* button to add about 5–10 orders, each with unique data.



The screenshot displays the MongoDB Atlas 'Collections' tab for a project. The left sidebar shows the database 'cst339' and the collection 'orders'. The main panel shows the 'cst339.orders' collection with a size of 587B, 5 total documents, and 36KB of indexes. Below this, there are tabs for 'Find', 'Indexes', 'Schema Anti-Patterns', 'Aggregation', and 'Search'. A filter bar is present with the text '{\"filter\": \"example\"}'. The 'QUERY RESULTS 1-5 OF 5' section shows two sample documents:

```
{
  "_id": ObjectId("5ed3e1a0ef93ced36b04098"),
  "orderNo": "0000000001",
  "productName": "This is Product 1",
  "price": 1.00,
  "quantity": 1
}
```

```
{
  "_id": ObjectId("5ed3e2e4ef93ced36b04099"),
  "orderNo": "0000000002",
  "productName": "This is Product 2",
  "price": 1.00,
  "quantity": 2
}
```



6. Create a User by selecting *Database Access* under the left pane. Click the Add a Database User button.
 - a. Select the Password Authentication Method.
 - b. Enter a Username and Password.
 - c. Set the Database User Privileges to Read and Write.
 - d. Click the *Add User* button.
7. Create a Network Access by selecting Network Access under the left pane. Click the Add IP Address button.
 - a. Click the *Allow Access From Anywhere* button (use this with caution).
 - b. Click the Confirm button.