

Table of Contents

Step 1: Map java class to database collection so that it becomes “DATABASE MANAGED ENTITY”	1
Step 3: MongoRepository	2
Step 4: service layer	2
Step 5: Custom method:.....	3
Step 6: OneToOne:.....	4
Step 7: Repo Layer:	5
Step 8: Service Layer:	5
Step 9: Controller Layer:.....	6

Step 1: Map java class to database collection so that it becomes “DATABASE MANAGED ENTITY”

1. To let spring data mongo template know how to map this class to a database collection and create the collection accordingly, update the Book class for respective annotations:

```
package com.boot.demo.sprinbootdemo.entity;

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

// tells to treat this class as database collection.
@Document(collection="books")
public class Book {
```

```

// Used for primary key identifier
@Id
// Used to map bookid property of java with _id as primary key of mongo db
@Field("_id")
private int bookid;
// rest remains the same
}

```

Step 3: MongoRepository

1. Spring boot provides an interface that takes care of basic CRUD operations
[GOOD NEWS!!! – NO NEED TO WRITE SQL QUERIES 😊]

Just extend the interface and we get the most relevant CRUD methods for standard data access available in a standard DAO.

```

package com.boot.demo.springbootdemo.repo;

import org.springframework.data.mongodb.repository.MongoRepository;
import java.util.List;

public interface BookRepo extends MongoRepository<Book, Integer> {}

```

2. MongoRepository<T,ID> is interface where
T : represents the database managed entity name
ID: represents the type of identifier.

Step 4: service layer

1. Create BookServiceRepo class to perform business operations on the book table as follows:

```

package com.boot.demo.springbootdemo.service;

import com.boot.demo.springbootdemo.entity.Book;
import com.boot.demo.springbootdemo.repo.BookRepo;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import java.util.List;

@Service
public class BookServiceRepo {
    @Autowired
    private BookRepo bookRepo;

    public long getTotalBookCount(){
        return bookRepo.count();
    }

    public List<Book> getAllBooks(){
        return bookRepo.findAll();
    }

    public Book addNewBook(Book book){
        if(bookRepo.existsById(book.getBookid()))
            throw new RuntimeException("Cannot add "+book.getBookid()+" already exists");
        return bookRepo.save(book);
    }

    public Book updateBook(Book book){
        if(!bookRepo.existsById(book.getBookid()))
            throw new RuntimeException ("cannot update "+book.getBookid()+" does not exist");
        return bookRepo.save(book);
    }

    public boolean deleteBook(int id){
        if(!bookRepo.existsById(id))
            throw new RuntimeException ("cannot delete "+id+" does not exist");
        bookRepo.deleteById(id);
        return true;
    }

    public List<Book> getBooksByAuthor(String author){
        return null;
    }
}

```

```

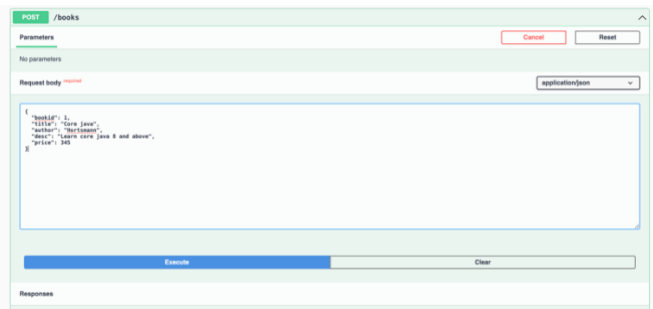
public Book getBookById(int id){
    if(!bookRepo.existsById(id))
        throw new RuntimeException (id+" not found");
    return bookRepo.findById(id).get();
}
}

```

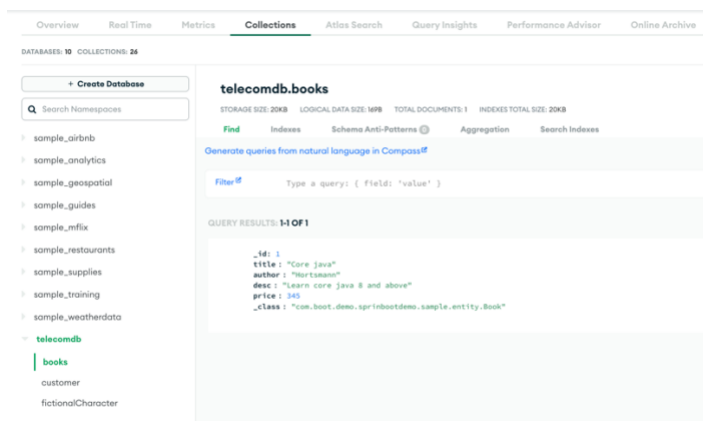
2. Just update the BookRestController class Autowired dependency from **BookService to BookServiceRepo**; everything should work as before. Just that this time it is communicating with a database and not stale data.

MAKE SURE TO EITHER ADD SOME DATA MANUALLY OR VIA POST REQUEST

3. From swagger when you make a post request to BookRestController api, it saves data in mongodb database as follows:



4. Check mongodb whether local or on cloud:



5. Likewise test for other methods as well.

Step 5: Custom method:

1. By default the MongoRepository provides all CRUD operations using the primary key. If we want to fetch data by a column other than primary key ex:
db.collection.find({author: " "})
2. Spring boot provides with conventions to follow for custom methods to execute custom queries:
<https://docs.spring.io/spring-data/mongodb/reference/mongodb/repositories/query-methods.html>
3. Just add below method in BookRepo file to fetch all books by author.
List<Book> findByAuthor(String author);
4. Update below method in BookServiceRepo class as follows:
public List<Book> getBooksByAuthor(String author){
 return bookRepo.findByAuthor(author);
}
5. Test using url http://localhost:8081/books?author=<authorname> and now you should get list of books by authors.

6. Apart from custom methods, you can also write mongodb queries. Make below changes:

- a. Add below in BookRepo:

```
@Query("{price : { $gt: ?0 } }") // SQL Equivalent : SELECT * FROM BOOK where price<=?
//@Query("{ price : { $gte: ?0 } }") // SQL Equivalent : SELECT * FROM BOOK where price>=?
//@Query("{ price : ?0 }") // SQL Equivalent : SELECT * FROM BOOK where price=?
List<Book> getBooksByPrice(Integer price);
```

- b. Update BookServiceRepo class by adding below method:

```
public List<Book> getBooksByPriceGreaterThan(int price) {
    return bookRepo.getBooksByPrice(price);
}
```

- c. Update BookRestController getBooks() method as follows:

```
@GetMapping
public List<Book> getBooks(@RequestParam(required = false) String author,
    @RequestParam(required = false) Integer price){
    logger.info("GET All books if author is null or get books by author : "+ author);
    if(author!=null )
        return bookService.getBooksByAuthor(author);
    if(price != null)
        return bookService.getBooksByPriceGreaterThan(price);
    return bookService.getAllBooks();
}
```

Step 6: OneToOne:

1. MongoDB database collections have no concept of relationships using primary and foreign key.
2. Create below 2 classes and you will notice @Document is only added on FictionalCharacter but not on Wand class as wand will be an object within FictionalCharacter class.

```
2.1. package com.boot.demo.springbootdemo.entity;
package com.boot.demo.springbootdemo.sample.entity;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import org.springframework.data.annotation.Id;
@Data
@AllArgsConstructor
@NoArgsConstructor
public class Wand {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;
    private String wood;
    private String core;
    private String length;
}
```

```
package com.boot.demo.springbootdemo.entity;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.mapping.Document;

@Document
@Data
@AllArgsConstructor
@NoArgsConstructor
public class FictionalCharacter {

    @Id
    private int id;
    private String name;
    private String house;
    private Wand wand;
```

```

private String bio;
private String imageUrl;
}

```

Step 7: Repo Layer:

1. Create Repos for character and wand as follows:

```

package com.boot.demo.springbootdemo.repo;

import com.boot.demo.springbootdemo.entity.Wand;
import com.boot.demo.springbootdemo.entity.Wand;
import org.springframework.data.mongodb.repository.MongoRepository;

public interface WandRepo extends JpaRepository<Wand, Integer> {
}

package com.boot.demo.springbootdemo.repo;

import com.boot.demo.springbootdemo.entity.FictionalCharacter;
import org.springframework.data.mongodb.repository.MongoRepository;

import java.util.List;

public interface CharacterRepo extends MongoRepository<FictionalCharacter,Integer > {

    public List<FictionalCharacter> findAllByHouse(String house);
    public FictionalCharacter findByName(String name);
    public boolean existsByName(String name);
}

```

Step 8: Service Layer:

1. Lets create CharacterService class for the business login layer.
2. **Update the class for spring specific annotations.**

```

package com.boot.demo.springbootdemo.service;

import com.boot.demo.springbootdemo.entity.FictionalCharacter;
import com.boot.demo.springbootdemo.entity.Wand;
import com.boot.demo.springbootdemo.repo.CharacterRepo;
import com.boot.demo.springbootdemo.repo.WandRepo;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.List;

public class CharacterService {

    private CharacterRepo characterRepo;

    public long getCount(){
        return this.characterRepo.count();
    }
    public List<FictionalCharacter> getCharacterByHouse(String house){
        return characterRepo.findAllByHouse(house);
    }
    public List<FictionalCharacter> getAllCharacters(){
        return characterRepo.findAll();
    }
    public FictionalCharacter getCharacterByName(String name){
        if(!characterRepo.existsByName(name))
            throw new RuntimeException(name+" not found");
        return characterRepo.findByName(name);
    }
    public FictionalCharacter getCharacterById(int id){
        if(!characterRepo.existsById(id))
            throw new RuntimeException(id+" not found");
        return characterRepo.findById(id).get();
    }
    public FictionalCharacter addNewCharacter(FictionalCharacter character){
        if(characterRepo.existsById(character.getId()))

```

```

        throw new RuntimeException("cannot insert "+character.getId()+" already exists");
        return characterRepo.save(character);
    }
    public FictionalCharacter updateCharacter(FictionalCharacter character){
        if(!characterRepo.existsById(character.getId()))
            throw new RuntimeException("cannot update "+character.getName()+" does not exist");
        return characterRepo.save(character);
    }
    public void deleteCharacter(int id){
        if(!characterRepo.existsById(id))
            throw new RuntimeException("cannot delete "+id+" does not exist");
        FictionalCharacter character = characterRepo.findById(id)
            .orElseThrow()->new RuntimeException("Could not delete character");
        characterRepo.delete(character);
    }
}

```

Step 9: Controller Layer:

1. Let create CharacterController class for the REST API endpoints and here you will also be introduced to logger.
2. `package com.boot.demo.springbootdemo.rest;`

```

import com.boot.demo.springbootdemo.entity.FictionalCharacter;
import com.boot.demo.springbootdemo.service.CharacterService;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.HashMap;
import java.util.List;
import java.util.Map;

@RestController
@RequestMapping("/api/characters")
@CrossOrigin(origins = "*", methods = {RequestMethod.DELETE, RequestMethod.GET, RequestMethod.PUT,
RequestMethod.POST})
public class CharacterController {

    Logger logger = LoggerFactory.getLogger(CharacterController.class);
    @Autowired
    private CharacterService characterService;

    @GetMapping
    public List<FictionalCharacter> getAllCharacters(@RequestParam(defaultValue = "all") String house){
        logger.info("GET All Characters for house {}", house);
        if(house.equals("all")) {
            List<FictionalCharacter> ob = characterService.getAllCharacters();
            System.out.println(ob.size());
            return ob;
        }else{
            List<FictionalCharacter> ob = characterService.getCharacterByHouse(house);
            System.out.println(ob.size());
            return ob;
        }
    }

    @GetMapping("/{id}/{id}")
    public ResponseEntity<Object> getCharacterById(@PathVariable int id){
        try {
            FictionalCharacter ob = characterService.getCharacterById(id);
            return ResponseEntity.ok(ob);
        }catch (RuntimeException e){
            Map<String, String> errorMap = new HashMap<>();
            errorMap.put("error", e.getMessage());
            return ResponseEntity.badRequest().body(errorMap);
        }
    }

    @GetMapping("/name/{name}")
    public ResponseEntity<Object> getCharacterByName(@PathVariable String name){
        try {
            FictionalCharacter ob = characterService.getCharacterByName(name);
            return ResponseEntity.ok(ob);
        }catch (RuntimeException e){

```

```

        Map<String, String> errorMap = new HashMap<>();
        errorMap.put("error", e.getMessage());
        return ResponseEntity.badRequest().body(errorMap);
    }
}

@PutMapping
public ResponseEntity<Object> updateCharacter(@RequestBody FictionalCharacter character){
    try {
        FictionalCharacter ob = characterService.updateCharacter(character);
        return ResponseEntity.ok(ob);
    } catch (RuntimeException e){
        Map<String, String> errorMap = new HashMap<>();
        errorMap.put("error", e.getMessage());
        return ResponseEntity.badRequest().body(errorMap);
    }
}

@DeleteMapping("/{id}")
public ResponseEntity<Object> deleteCharacter(@PathVariable int id){
    Map<String, String> map = new HashMap<>();
    try {
        characterService.deleteCharacter(id);
        map.put("message", "Delete successful");
        return ResponseEntity.ok(map);
    } catch (RuntimeException e){
        map.put("error", e.getMessage());
        return ResponseEntity.badRequest().body(map);
    }
}

@PostMapping()
public ResponseEntity<Object> addCharacter(@RequestBody FictionalCharacter character){
    try {
        FictionalCharacter ob = characterService.addNewCharacter(character);
        return ResponseEntity.ok(ob);
    } catch (RuntimeException e){
        Map<String, String> errorMap = new HashMap<>();
        errorMap.put("error", e.getMessage());
        return ResponseEntity.badRequest().body(errorMap);
    }
}
}

```

3. Please test the above API from swagger.